



Calhoun: The NPS Institutional Archive
DSpace Repository

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

1978

A serial fiber optic data bus for a distributed microcomputer system.

Schue, Richard Ernest

Monterey, California. Naval Postgraduate School

<http://hdl.handle.net/10945/18543>

Downloaded from NPS Archive: Calhoun



Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

<http://www.nps.edu/library>

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93940

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

A Serial Fiber Optic Data Bus
for a
Distributed Microcomputer System

by

Richard Ernest Schue

September 1978

Thesis Advisor:

U. R. Kodres

Approved for public release; distribution unlimited

T187398

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS
BEFORE COMPLETING FORM

1. REPORT NUMBER		2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Serial Fiber Optic Data Bus for a Distributed Microcomputer System			5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; September 1978
7. AUTHOR(s) Richard Ernest Schue			6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940			8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940			10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940			12. REPORT DATE September 1978
			13. NUMBER OF PAGES 196
			15. SECURITY CLASS. (of this report) Unclassified
			15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited			
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)			
18. SUPPLEMENTARY NOTES			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Serial Data Bus -- Fiber Optics -- Microcomputers -- Distributed Processing			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The design and development of a serial fiber optic data bus to connect several Intel single board computers in a distributed processing system is described in this thesis. Included are details on the serial bus control and message handling techniques developed for the system. Also included is a complete description of the hardware that was constructed. Commercially available fiber optic components were used in the data bus. The interface software is compatible with a real-time operating system previously developed at the Naval Postgraduate School.			

A Serial Fiber Optic Data Bus
for a
Distributed Microcomputer System

by

Richard Ernest Schue
Naval Avionics Center
B.S., Rose-Hulman Institute of Technology, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1978

1.1
1.2
1.3

ABSTRACT

The design and development of a serial fiber optic data bus to connect several Intel single board computers in a distributed processing system is described in this thesis. Included are details on the serial bus control and message handling techniques developed for the system. Also included is a complete description of the hardware that was constructed. Commercially available fiber optic components were used in the data bus. The interface software is compatible with a real-time operating system previously developed at the Naval Postgraduate School.

TABLE OF CONTENTS

I.	INTRODUCTION	10
II.	BACKGROUND	12
	A. MILITARY APPLICATIONS OF FIBER OPTICS . . .	12
	B. MILITARY SERIAL DATA BUS STANDARD	13
	C. THE DISTRIBUTED COMPUTER CONCEPT	14
	D. THE DISTRIBUTED MICROCOMPUTER SYSTEM	17
	E. THE SERIAL INTERFACE	18
III.	SYSTEM DESIGN	20
	A. BASIC DESIGN PHILOSOPHY	20
	B. THE SERIAL BUS CONTROL PROBLEM	21
	C. MESSAGE HANDLING IN THE OPERATING SYSTEM . .	24
	D. CORPORATE BUFFER OPERATION	30
	E. SERIAL MESSAGE HANDLING TECHNIQUES	31
	F. SERIAL INTERFACE TERMINAL OPERATION	37
IV.	HARDWARE DESIGN	42
	A. SERIAL BUS INTERFACE	42
	1. Transmitter	42
	2. Receiver	46
	3. Clock Synchronizer	46
	4. Sync Detector	49
	5. Controller	51
	B. PARALLEL BUS INTERFACE	51
	C. CPU GROUP	52
	D. ELECTRO-OPTIC COMPONENTS	52
	E. CONSTRUCTION DETAILS	53

V.	SOFTWARE DESIGN	55
	A. PARALLEL MESSAGE HANDLING ROUTINES	55
	B. SERIAL MESSAGE HANDLING ROUTINES	56
VI.	SYSTEM INTEGRATION	59
VII.	CONCLUSIONS	61
APPENDIX A	HARDWARE REFERENCE MANUAL	63
APPENDIX B	SOFTWARE REFERENCE MANUAL	105
APPENDIX C	PROGRAM LISTING	147
APPENDIX D	DEBUGGING TECHNIQUES	182
BIBLIOGRAPHY	195
INITIAL DISTRIBUTION LIST	196

LIST OF TABLES

Table 1:	Serial Interface Controller ROM Program	84
Table 2:	Serial Bus Terminal States	141
Table 3:	Serial Bus Command Codes	142
Table 4:	Corporate Memory Buffer and Control Registers	143
Table 5:	CPU Control Lines	144
Table 6:	CPU Latch Controls	145
Table 7:	8748 Memory Allocation	146
Table 8:	8748 Debug Switch Settings	193
Table 9:	Interface Cable Wiring	194
Table 10:	I/O Port Control Words	194

LIST OF DRAWINGS

Figure 1:	MIL-STD-1553 Word Formats	15
Figure 2:	MIL-STD-1553 Message Formats	16
Figure 3:	A Distributed Microcomputer System	19
Figure 4:	Serial Message Formats	25
Figure 5:	Corporate Message Buffer	27
Figure 6:	A Message in the Corporate Buffer	29
Figure 7:	Corporate Buffer Operation	32
Figure 8:	8748 Message Buffer	34
Figure 9:	8748 Buffer Operation	36
Figure 10:	Serial Interface Terminal States	39
Figure 11:	Serial Interface Block Diagram	43
Figure 12:	Transmitter and Receiver Block Diagrams	44
Figure 13:	Clock Synchronizer Operation	48
Figure 14:	Sync Waveforms	50
Figure 15:	Optical Data Bus Configuration	54
Figure 16:	Clock Sync Schematic	67
Figure 17:	Clock Sync Timing	68
Figure 18:	Sync Detector Schematic	72
Figure 19:	Sync Detector Transmit Mode Timing	73
Figure 20:	Sync Detector Receive Timing - Data Sync	74
Figure 21:	Sync Detector Receive Timing - Command Sync, Leading 0	75
Figure 22:	Sync Detector Receive Timing - Command Sync, Leading 1	76
Figure 23:	Controller Schematic	79

Figure 24:	Controller Receive Sequence	80
Figure 25:	Controller Transmit Sequence - Data Sync (Part 1)	81
Figure 26:	Controller Transmit Sequence - Data Sync (Part 2)	82
Figure 27:	Controller Transmit Sequence - Command Sync	83
Figure 28:	Transmitter Schematic	89
Figure 29:	Receiver Schematic	92
Figure 30:	Parallel Bus Interface Schematic	95
Figure 31:	Multibus Controller Schematic	96
Figure 32:	Multibus Controller Timing	97
Figure 33:	CPU Schematic (Part 1)	102
Figure 34:	CPU Schematic (Part 2)	103
Figure 35:	Component Placement	104
Figure 36:	Time Out Routine Flow Chart	106
Figure 37:	Restart Routine Flow Chart	109
Figure 38:	Executive Routine Flow Chart	112
Figure 39:	Receive External Message Routine Flow Chart	116
Figure 40:	Send External Message Routine Flow Chart	119
Figure 41:	Set EXTMSG Routine Flow Chart	123
Figure 42:	Message Buffer Lock Routine Flow Chart	124
Figure 43:	Serial Data Message Receive Routine Flow Chart	127
Figure 44:	Serial Message Decoder Routine Flow Chart	129
Figure 45:	Request-to-Send Command Service Routine Flow Chart	131

Figure 46:	End-of-Message Command Service Routine Flow Chart	133
Figure 47:	Bus Grant Command Service Routine Flow Chart	135
Figure 48:	Clear-to-Send Acknowledgement Routine Flow Chart	137
Figure 49:	End-of-Message Acknowledgement Routine Flow Chart	137
Figure 50:	Bus Grant Acknowledgement Routine Flow Chart	137
Figure 51:	Data Message Acknowledgement Routine Flow Chart	138
Figure 52:	Interrupt Return Routine Flow Chart . . .	140

I. INTRODUCTION

Recent advances in the field of fiber optics have made this technology a candidate for present and future military data transmission systems. Fiber optic data links provide wide bandwidth, low attenuation, and excellent resistance to electromagnetic interference.

The feasibility of the serial data bus has been demonstrated through applications of MIL-STD-1553 in the F16, F18 and other military platforms. The MIL-STD-1553 provides time multiplexed data transmission between a central computer and a number of terminals, replacing the individual point-to-point hardwired interface connections with a single twisted pair cable.

The introduction of small single board computers with considerable computational power has made the distributed system of microcomputers a viable candidate to replace the second generation computers now in use in military tactical data systems. Such a system offers reductions in costs, weight and power consumption with increased reliability and flexibility.

These three developments led to the investigation of a serial fiber optic data bus for a distributed microcomputer system. Such a data bus allows several microcomputers to be joined into a single system although they may be physically separated by some distance.

The project involved the design and construction of the hardware used to interface the computers with the serial data bus and the development of the software required for bus control and message handling. The software was designed around a real-time operating system previously developed at the Naval Postgraduate School for use in such a distributed system. Commercially manufactured fiber optic components were utilized in the serial bus.

The serial bus control problem is first examined and the bus protocol used in this system is outlined. Message handling procedures used in the operating system are described and similarities are drawn with the message handling employed in the serial bus. The interface hardware and software are discussed briefly with details presented in the appendices. Debugging techniques along with the complete program listing also appear in the appendices.

II. BACKGROUND

A. MILITARY APPLICATIONS OF FIBER OPTICS

The optical transmission of digital data avoids many of the limitations of the traditional hardwired methods. Since the transmission medium is a dielectric, fiber optic cables provide total isolation between transmitter and receiver. Crosstalk between adjacent lines is easily eliminated by surrounding each cable with an opaque jacket. Fiber optics neither generate nor are subject to electromagnetic interference, EMI. Optical transmission lines experience none of the ringing problems associated with wire systems, and provide greater bandwidth with reduced attenuation.

Commercial applications of fiber optics have been mainly in the field of telecommunications, where the wider bandwidth and reduced attenuation of the optical cable is desired. Military uses of fiber optics frequently involve short run lengths, under 150 feet, and bandwidth requirements of less than 10 Mbits/sec. The advantages of fiber optics in military systems is in their EMI and electromagnetic pulse (EMP) characteristics and in their weight savings over a comparably shielded coaxial cable. In addition, the wide bandwidth capability of fiber optic transmission lines provides room for expansion as tactical data systems become more complex. The high temperature capability of glass and fused silica fibers

allow cables to be used in extreme environmental conditions, as in the engine compartments of jet aircraft.

The Navy's Airborne Light Optical Fiber Technology (ALOFT) project is a prominent demonstration of the application of fiber optic technology in the tactical environment. ALOFT involved the replacement of 150 hardwired interface connections aboard an A-7 aircraft with 12 digital and one analog fiber optic communication links. The 12 digital links form six bidirectional communication lines between a central computer and five remote terminals. The data rate was 50Kbits/sec. The longest cable run was about 27 feet. Some cables required as many as five optical couplings to complete their path through bulkheads and the like. Subsequently the losses incurred in the couplings were the major source of optical attenuation with the fiber attenuation a secondary contributor.

B. MILITARY SERIAL DATA BUS STANDARD

The military standard for serial time-division multiplexed data buses is MIL-STD-1553A. MIL-STD-1553A is a command/response data bus which utilizes a central bus controller to preside over all bus transactions. All other terminals in the system respond only to commands from the controller and do not initiate transmissions on their own.

The bus consists of a single twisted pair cable. Data is time-division multiplexed in an asynchronous, half

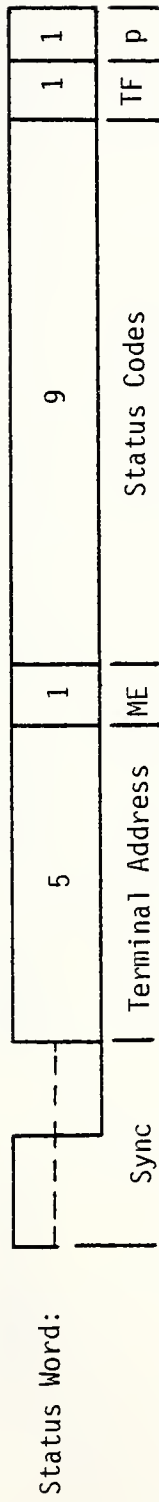
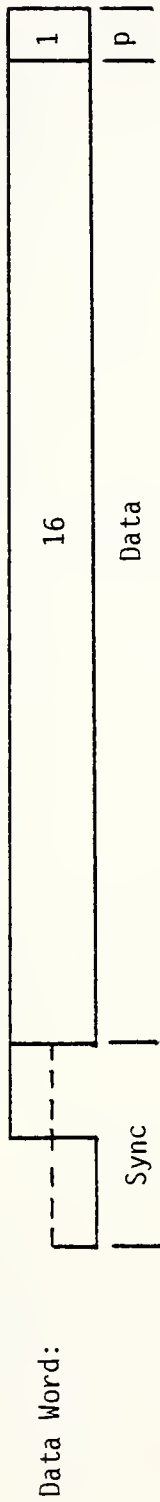
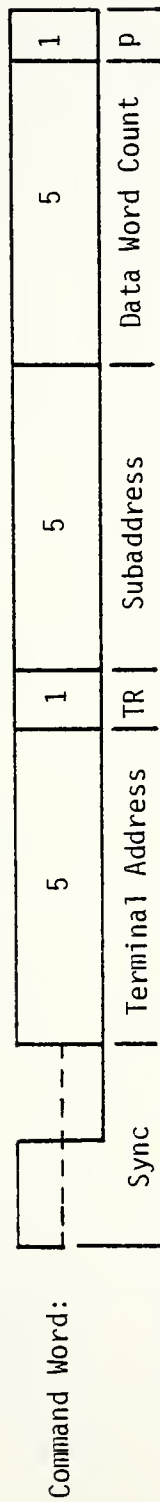
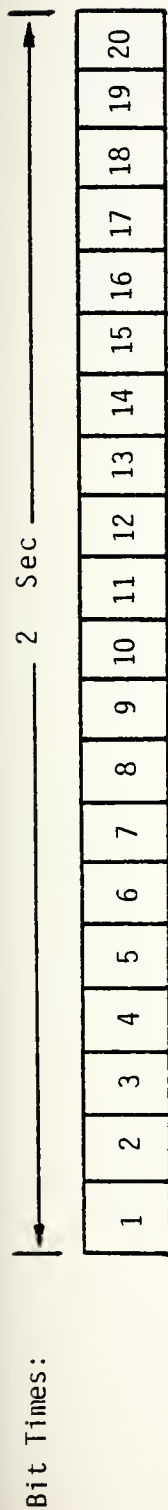
duplex manner. Data is encoded in bi-phase Manchester code and transmitted at 1 Mbit/sec. Word size is fixed at 20 bits per word, consisting of 16 bits of data, 1 parity bit and three sync bits. Word formats are shown in figure 1.

Bus traffic is composed of three types of messages; command words, status words, and data words. Command words are sent from the bus controller to the terminals to instruct them when to transmit and receive. Status words are sent to the controller from the terminal in response to a command word, and data words form the actual data transfer. The message formats are shown in figure 2. MIL-STD-1553 has been proven to work in the field and is adequate for many applications where a central computer is controlling a number of peripheral terminals.

C. THE DISTRIBUTED COMPUTER CONCEPT

The distributed computer concept grew out of the desire to make a number of small computers do the job of a larger computer. It differs from the more conventional centralized computer in that several processors are working on more or less unrelated tasks at the same time. System resources and data banks are accessed through a common or corporate memory serving all the computers.

The distributed system may use standardized computer modules lowering system costs and enhancing maintainability. Commercially available microprocessors, with their excellent



TR: Transmit/Receive, P: Parity, ME: Message Error, TF: Terminal Flag

Figure 1: MIL-STD-1553 Word Formats

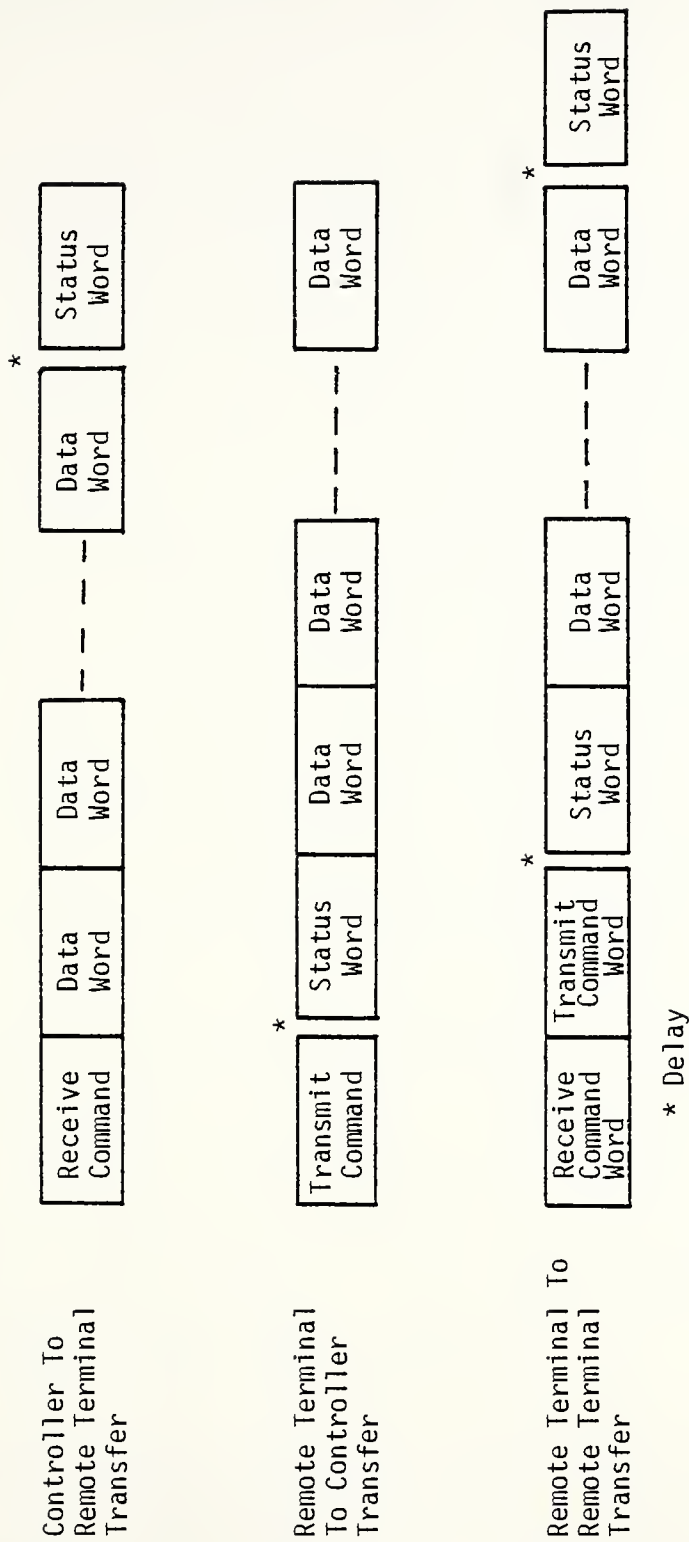


Figure 2: MIL-STD-1553 Message Formats

reliability record would be a worthy component in such a system.

D. THE DISTRIBUTED MICROCOMPUTER SYSTEM

The distributed computer system prototype developed at the Naval Postgraduate School centers around the use of the Intel SBC 80/20-04 single board computer. The SBC 80/20-04 is representative of the current technology in single board computers at the time of this writing. This is an 8080 based system with 4096 bytes of read-write memory and 8192 bytes of read-only memory on board. The unit provides 48 programmable I/O lines and eight programmable interrupt inputs. The board supports an external parallel bus, the Intel multibus, which allows expansion to off-board memory and I/O facilities. One or more of these computers along with corporate read/write memory form an affinity group.

The real-time operating system developed for this system uses a modular program structure. A module is a software program to handle a specific function or a group of functions. Modules can be written and compiled separately allowing easy modification of the software. Up to eight modules may reside in an individual single board computer.

Modules are given consecutive numbers and communicate with each other through messages passed from module to module. If two modules reside in different computers, the messages are passed by way of a dedicated area in common memory.

Each computer may work on several tasks at once, accessing the common resources in corporate memory as required. Each computer has a resident operating system in on-board read-only memory. Data that is used often by the computer is stored in local read-write memory to improve access time and reduce the external bus use.

E. THE SERIAL INTERFACE

The serial interface described in this thesis provides a means of linking single board computers at physically separated locations with a fiber optic data bus to form a distributed computer system, as illustrated in figure 3. The distributed system becomes a collection of individual stations each having one or more single board computers, corporate memory, and a serial interface. The serial interface transfers messages from corporate memory at one location to corporate memory at a remote location. The interface unit contains an Intel 8748 single chip computer to handle the serial bus protocol. No special additional requirements are placed on the operating system because of the serial bus.

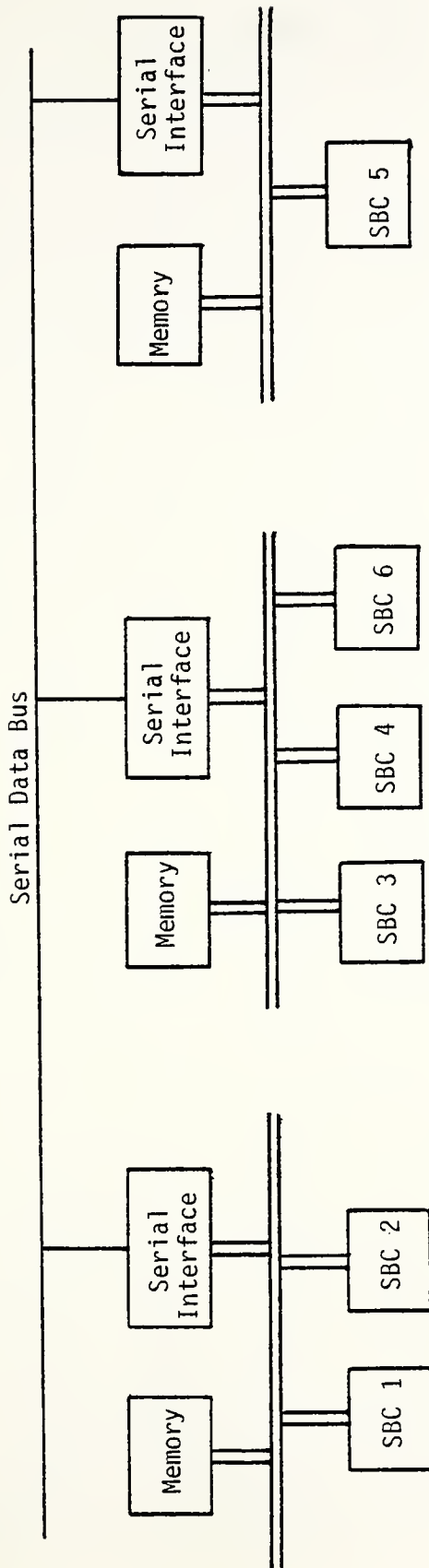


Figure 3: A Distributed Microcomputer System

III. SYSTEM DESIGN

A. BASIC DESIGN PHILOSOPHY

The serial bus described in this thesis was viewed from the beginning as an extension of the parallel Intel Multibus. This was done mainly to make the serial link between affinity groups transparent to the operating system. In the Multibus system, the single board computers communicate with one another by way of messages deposited in corporate memory. The messages can be viewed as letters, the corporate memory as the mailboxes, and the parallel bus as the mail carrier. Each single board computer periodically checks its mailbox to receive messages from other computers.

The function of the serial data bus is to link several affinity groups as if they were all on the same parallel bus. Messages are sent from computer to computer in the same manner as before. A serial interface terminal resides on each parallel bus and scans the corporate memory for messages bound for non-local computers. When such a message is detected, the message is sent via the serial bus to the terminal on the destination computer's parallel bus. The message is then deposited in corporate memory at the destination where it can be serviced by the computer. In this manner, a computer can communicate with any other in the system without changing the message handling procedures used to communicate with local computers.

B. THE SERIAL BUS CONTROL PROBLEM

Communicating via a single wire, or fiber optic cable, presents some additional problems in bus control when contrasted with a parallel bus. In a parallel bus, separate signaling lines are provided. In the Multibus, each single board computer has a bus request line going to the central bus control arbiter and a bus grant line coming from the controller. Since signaling is done on separate lines, it does not interfere with data transmission.

In the single wire bus, signaling must be done on the same line as the message sending. Some of the messages sent on the serial bus must be bus control messages from terminal to terminal. The military standard for serial time-division multiplexed data buses is MIL-STD-1553. This standard has served as a guide for serial data buses now in use in tactical data systems, and was examined as a possibility for this project.

The major drawbacks of the MIL-STD-1553 bus for use in a distributed processing system are incurred by the single bus controller and the limitations on the remote terminals. In the serial bus design presented herein, the bus control is decentralized, allowing each terminal to send messages under its own control. This design provides the terminals with equal communication power needed for a distributed computer system.

The terminals in this design access the bus through a 'round-robin' type of bus control. This is a simple method of passing control from one terminal to the next, insuring that no two terminals try to access the bus at the same time. The computers in the system are numbered, starting from one, to the maximum number in the system. The serial interface terminal serves all computers on its local bus. The terminal reads the local parallel bus roster by means of a dedicated location in memory. The terminals also read their terminal identification numbers and the identification number of the next terminal to receive bus control. The terminal with identification number one becomes the initial bus control station.

The initial bus control terminal checks for any outgoing messages waiting to be sent over the serial bus. If none are waiting, the terminal issues a bus grant command to the next terminal to receive the bus control. This process continues until the last terminal, the one with the numerically highest identification number returns control to terminal number one.

Messages are preceded by a request-to-send command to the desired terminal. The command is addressed to the destination computer number. The receiving terminals scan their individual bus roster to determine if that computer number resides on their bus. The request-to-send command also contains the message length. If the destination terminal can accommodate the message, it replies with an acknowledgement. If insufficient space is available, no

reply is made and the requesting terminal senses a time-out. When this occurs, the requesting terminal turns the bus over to the next terminal in line and waits until it can send the message again.

After receiving a 'clear-to-send' acknowledgement, the requesting terminal sends data messages one byte at a time to the destination terminal. After receiving each data message, the receiving terminal responds with a reply. The sending terminal waits for a reply before sending the next data message. If no reply is received within a certain time, the message is repeated. If parity errors are detected in the received message, no reply is issued and the sender automatically repeats the message. At the end of the message, the sending terminal issues an end-of-message command. The receiving terminal returns to a listener mode, and the sending terminal grants the bus to the next terminal.

Command messages are provided for the following functions:

- Bus Grant

- Request-to-Send

- End-of-Message

Acknowledgements are issued for:

- Bus Grant Acknowledge

- Clear-to-Send Acknowledge

- End-of-Message Acknowledge

- Data Acknowledge

The command messages consist of three bit intervals of sync character, four bits of destination address, four bits of origin address, eight bits of command information, and one parity bit for a total of 20 bits per message. The data messages follow a similar format with a different sync character at the beginning and carry one byte of data per message. The address fields limit the system to 15 computers in the present configuration. The message formats are controlled by firmware and are easily changed to accommodate different message lengths. This could allow for sixteen bits of data and/or eight bit address fields to handle larger systems. Message formats are shown in figure 4.

Acknowledgements take no special message format. The sending terminal treats any received message as an acknowledgement. This eliminates the possibility of parity errors requiring an acknowledge message to be repeated. The acknowledgement is essentially error resistant.

C. MESSAGE HANDLING IN THE OPERATING SYSTEM

Messages passed from module to mode are a minimum of four and a maximum of 36 bytes in length. All messages contain a header consisting of the receiving module number (RMN), the sending module number (SMN), the message number (MN), and the message length (ML). The rest of the message is data and may contain zero to 32 bytes. Message numbers are agreed upon by the modules involved

Data Message:

D	3	2	1	0	3	2	1	0	7	6	5	4	3	2	1	0	P
Data Sync	Destination SBC				Source SBC				Data Byte								Parity Bit

Command Message:

C	3	2	1	0	3	2	1	0	7	6	5	4	3	2	1	0	P
Command Sync	Destination SBC				Source SBC				Command Byte								Parity Bit

Command Bytes:

Bit	7	6	5	4	3	2	1	0	
	L ₅	L ₄	L ₃	L ₂	L ₁	0	0	0	Request-to-Send (Message Length=L ₅ L ₄ L ₃ L ₂ L ₁ 0)
	0	0	0	0	0	0	0	1	End-of-Message
	0	0	0	0	0	0	1	0	Bus Grant

Figure 4: Serial Message Formats

and are useful for sending control messages where no data is passed. Module numbers are unique to each module. A maximum of eight modules per single board computer is allowed in the present version of the operating system. Modules number zero through seven reside in computer number one, (SBC number 1), modules eight through 15 in SBC number two and so forth.

The message buffer employed in the operating system is a linear buffer of 248 bytes in length located from F022 (hex) to F119 (hex) in corporate memory. Six additional memory locations are used as control registers, as shown in figure 5. The use of these registers is described herewith.

EXTMSGLOCK

This is the lock mechanism for the corporate buffer. If none of the local single board computers are using the buffer, the buffer is termed 'unlocked' and EXTMSGLOCK is set to zero. When one of the computers is using the buffer, EXTMSGLOCK is set equal to the SBC number of that computer. The lock mechanism prevents more than one computer from accessing the buffer at any one time.

EXTMSG

This register indicates the SBC number of the computer to receive the next outgoing message. If the buffer is empty EXTMSG equals zero. EXTMSG is set by the last

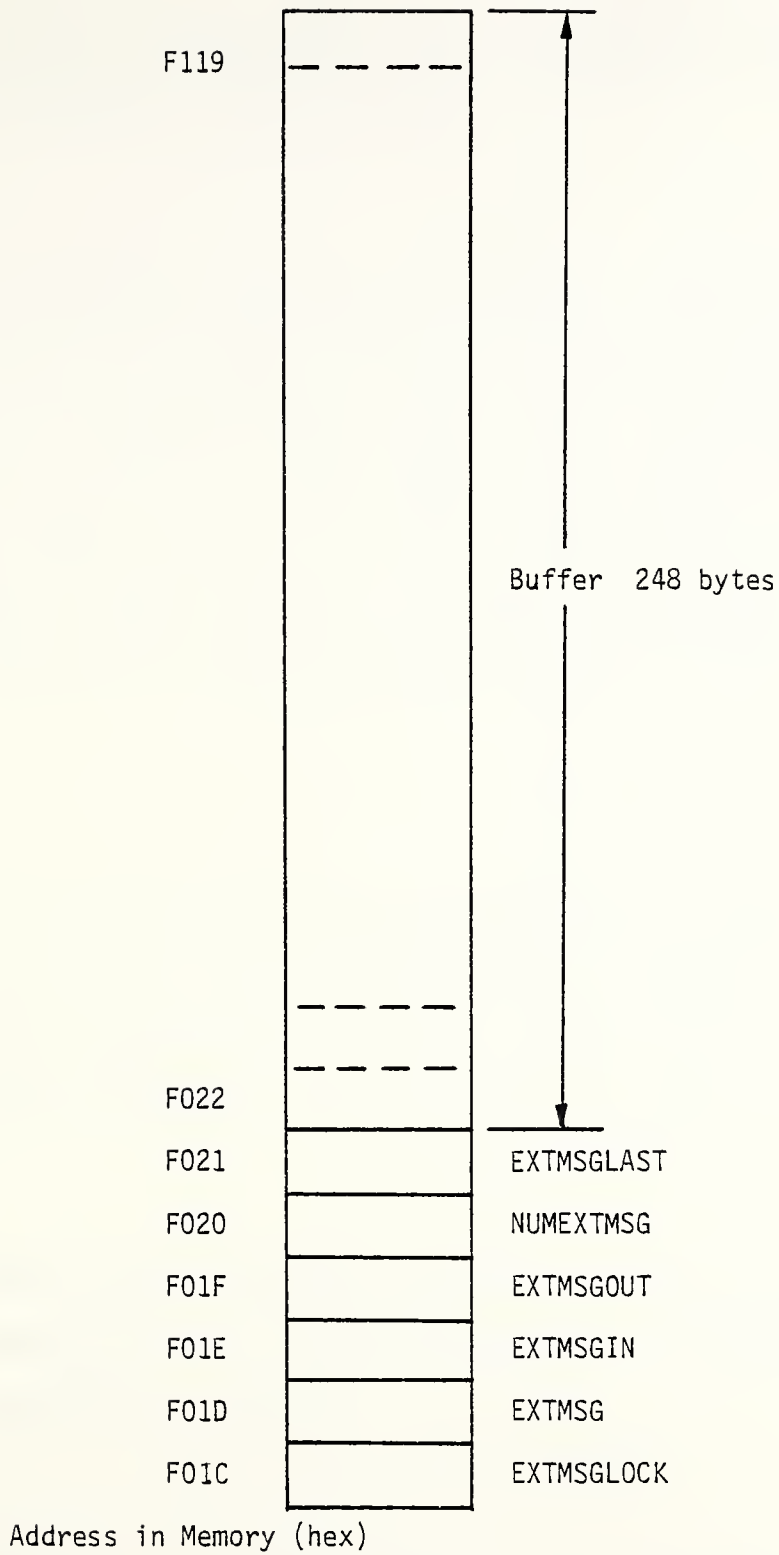


Figure 5: Corporate Message Buffer

computer that uses the buffer, and is periodically scanned by all the computers in the system.

EXTMSGIN

This is the input pointer of the buffer. EXTMSGIN indicates the next available location in memory, and is incremented by the sender after each byte is transferred. The pointer is relative to the bottom of the buffer and can assume values from 0 to F7 (hex). The address of the next available memory location is found by adding the pointer value to the base address of the buffer, F022 (hex).

EXTMSGOUT

This is the corporate buffer output pointer and it functions similarly to the input pointer. EXTMSGOUT is also expressed relative to the bottom of the buffer. The use of the pointers is illustrated in figure 6.

NUMEXTMSG

The value at this location in memory indicates the number of messages in the corporate buffer. If NUMEXTMSG is zero, the buffer is empty.

EXTMSGLAST

This register is used as a flag to mark the end of the current series of messages in memory. When an

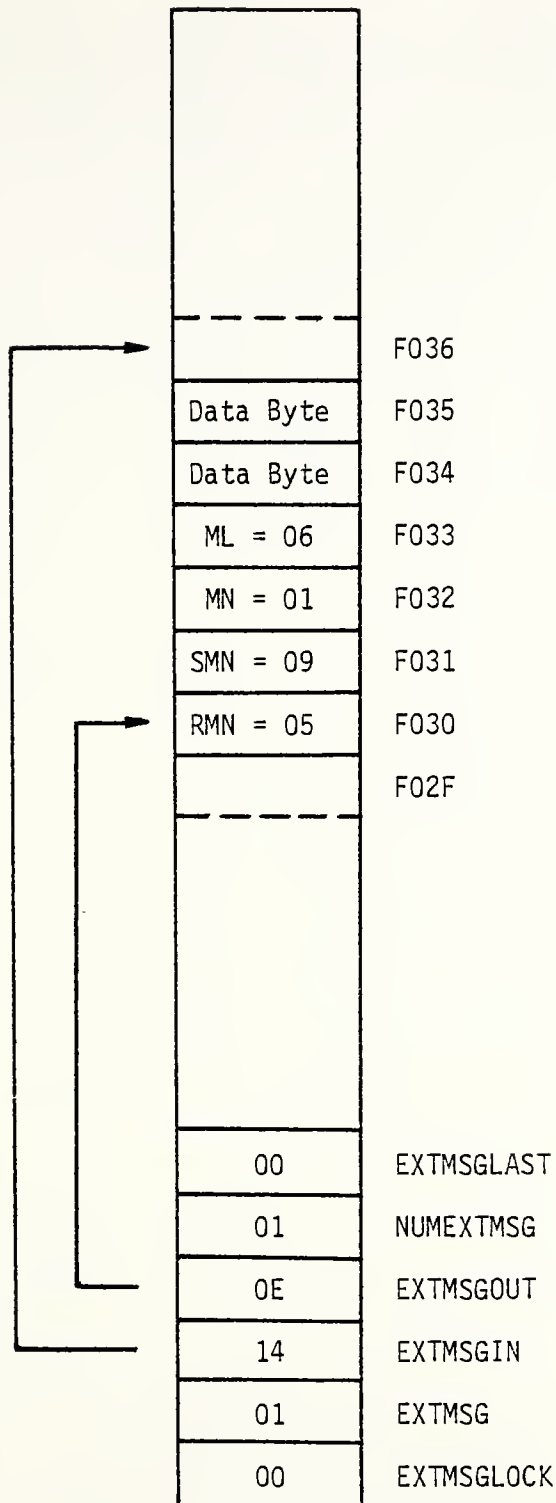


Figure 6: A Message in the Corporate Buffer

incoming message does not fit into the remaining room at the top of the buffer, EXTMSGLAST is set equal to the last value of the input pointer. The input pointer is set to zero and the message is stored starting at the bottom of the buffer. When the last message at the top of the buffer has been removed, EXTMSGOUT will be equal to EXTMSGLAST, signaling the computer that no more messages remain at the top of the buffer and the next outgoing message begins at the bottom.

D. CORPORATE BUFFER OPERATION

Computers wishing to access the corporate buffer must wait until the current user has unlocked the buffer. This is accomplished by continually reading and re-reading the lock register until it is zero. When an unlocked buffer is detected, the waiting computer writes its SBC number into the lock. It then reads the lock to make sure another computer did not already lock the buffer.

After lock has been achieved, a computer wishing to deposit a message checks the buffer for available room. If the output pointer is greater than the input pointer, the available space lies between the two pointers. If the input pointer is above the output pointer, the empty buffer lies above the input pointer and below the output pointer. If the message does not fit into the top of the buffer, the input pointer is zero'ed, and EXTMSGLAST is

set to its previous value. The computer then attempts to fit the message into the room at the bottom of the buffer. If the message does not fit, the transaction is aborted, and a buffer overflow condition exists. Messages are not allowed to 'wrap around' the top of the buffer into empty locations at the bottom, as shown in figure 7.

Any time the buffer is empty, both pointers are reset to zero. A buffer overflow is detected if the input pointer is equal to the output pointer when the number messages in the buffer is not zero. NUMEXTMSG is incremented after each message is deposited and decremented after each withdrawal.

Each time a message is removed from the buffer, the computer in charge reads the receive module number (RMN) of the next message, computes the SBC number corresponding to that module, and loads the SBC number into EXTMSG. If the message just removed was the last message in the buffer, EXTMSG is set to zero. After a message is deposited into the buffer, EXTMSG is set only if the message just loaded is the only message in the buffer.

E. SERIAL MESSAGE HANDLING TECHNIQUES

The message handling techniques employed in the serial terminals are patterned after the message handler used in the operating system for the single board computers. Messages are stored in a circular first in-first out (FIFO)

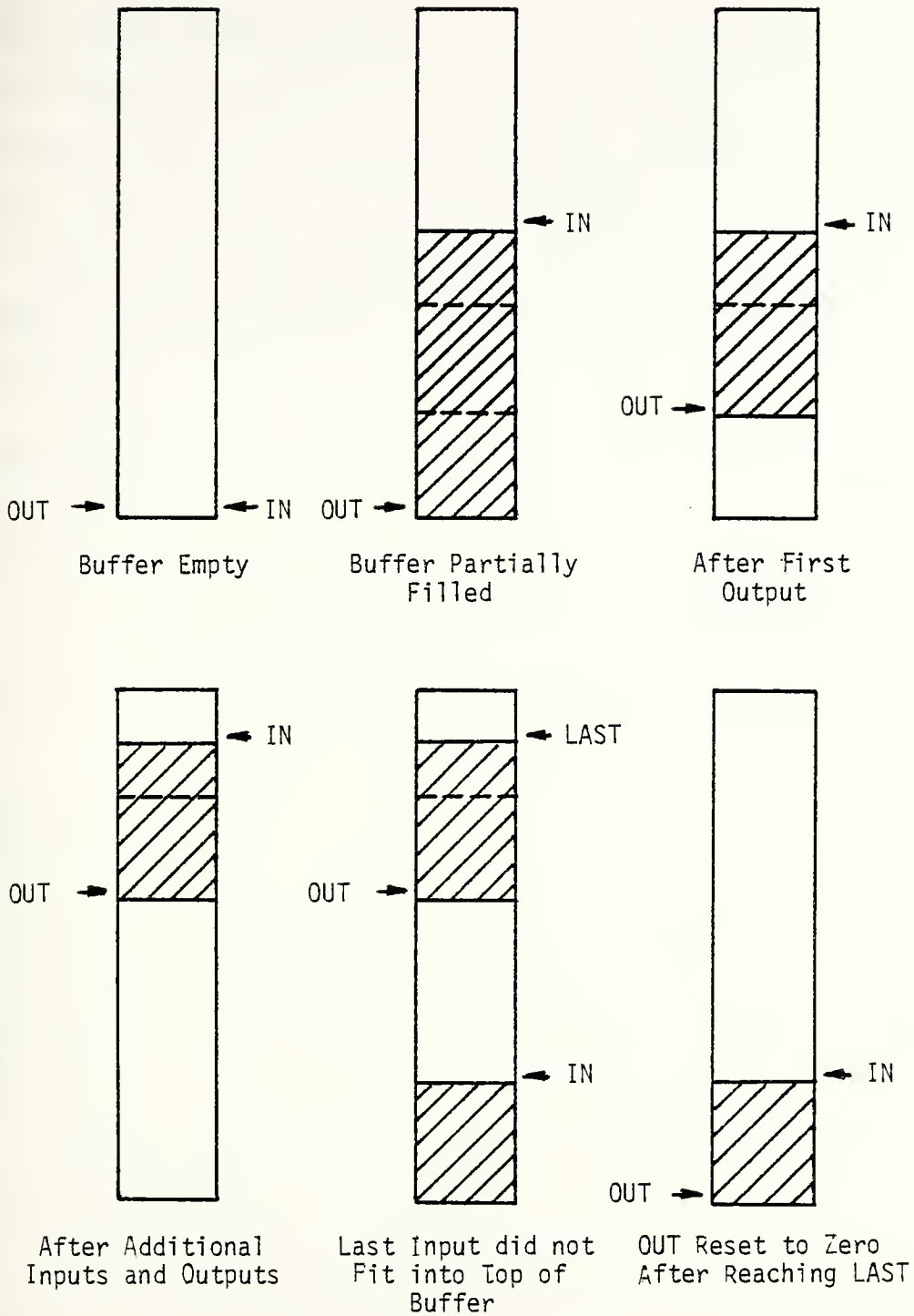


Figure 7: Corporate Buffer Operation

buffer of 32 bytes in length. But while the buffer in corporate memory allows only one user at a time, the buffer in the serial terminals can be accessed by two users at once. Thus a serial write can take place during a parallel read and vice versa. Seven additional registers control the operation of the buffer in the serial terminal and are shown in figure 8.

Input Pointer

Output Pointer

The pointers contain the address of the next outgoing byte and the next vacant memory location for incoming data. These contain the absolute address of the data in the buffer.

Input Counter

Output Counter

The input and output counters are not actually involved in the buffer control but are dedicated to be used to count the number of bytes transferred. The counters are loaded with the length of the message at the beginning of the transfer and decremented after each byte is moved. When the counters are zero, the transfer is complete.

Input Lock

Output Lock

The input and output locks are zero if no one is using the buffer and contain the SBC number of the computer

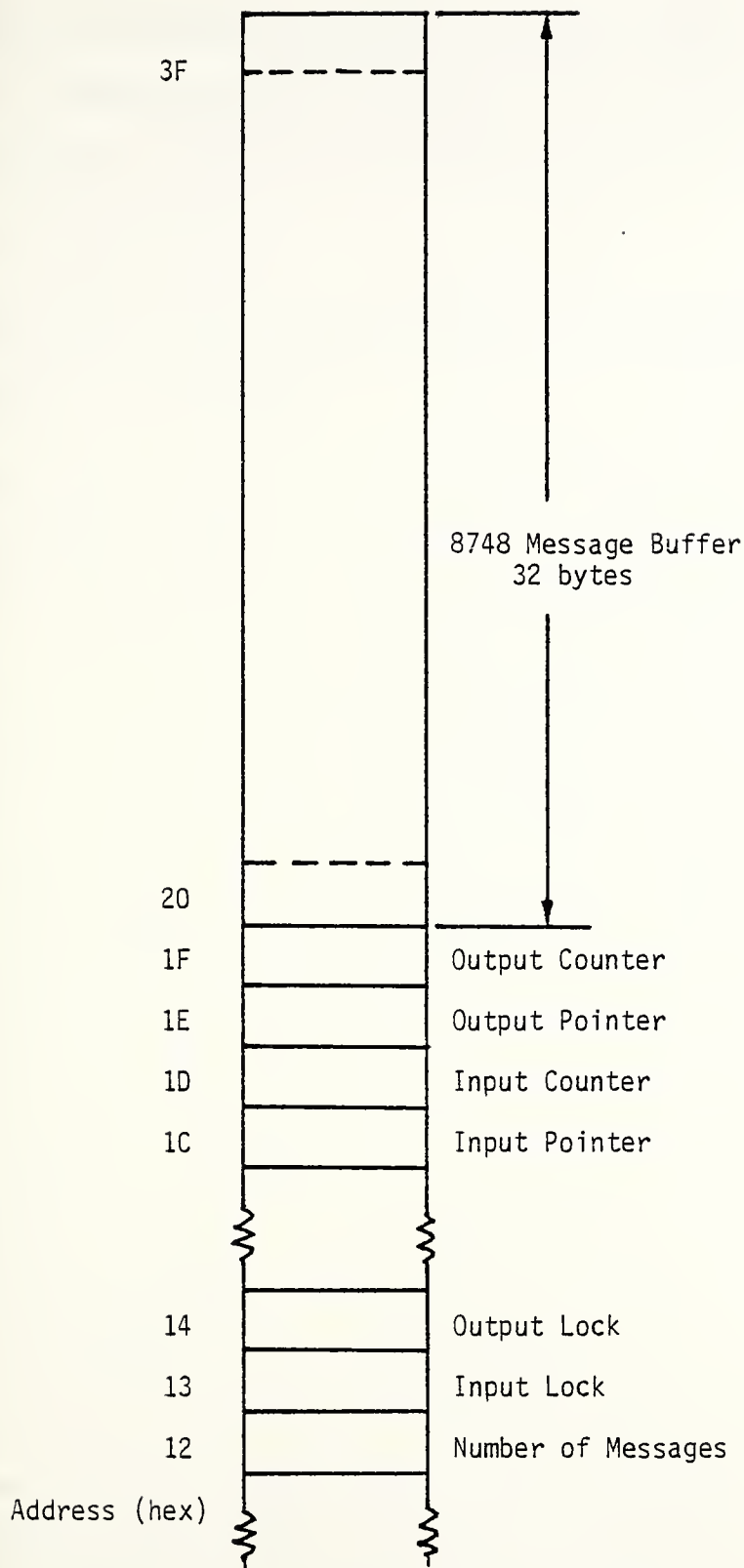


Figure 8: 8748 Message Buffer

currently using the buffer when a transaction is taking place. Separate input and output locks are provided to allow the read while write feature described earlier.

Number of Messages

This last register keeps track of the number of messages currently in the buffer.

The message buffer in the serial terminal is operated in a true circular fashion, in contrast to the linear buffer used in the corporate memory. This is done for two reasons: the first, to provide maximal utilization of the modest buffer size, and secondly, to allow a simpler buffer control algorithm. The input and output pointers are initially set to the bottom of the buffer during power-up. After receiving the first message, the input pointer is raised to the next vacant memory location, while the output pointer remains at the bottom of the buffer. As additional messages are received, via either the serial or parallel channels, the input pointer continues to be advanced. During an output operation, the output pointer is advanced as each byte is read from the buffer. When the input pointer reaches the top of the buffer, it is reset to the bottom of the buffer and the input operation continues, as shown in figure 9.

The output pointer follows in the same fashion, continually chasing the input pointer around the circular buffer. The available room in the buffer is easily

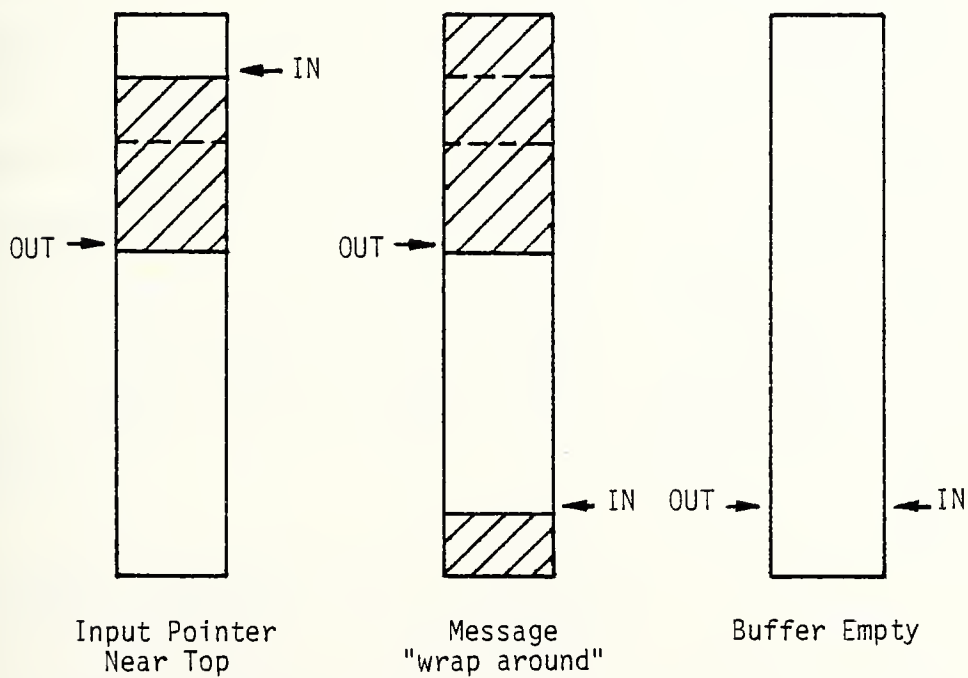
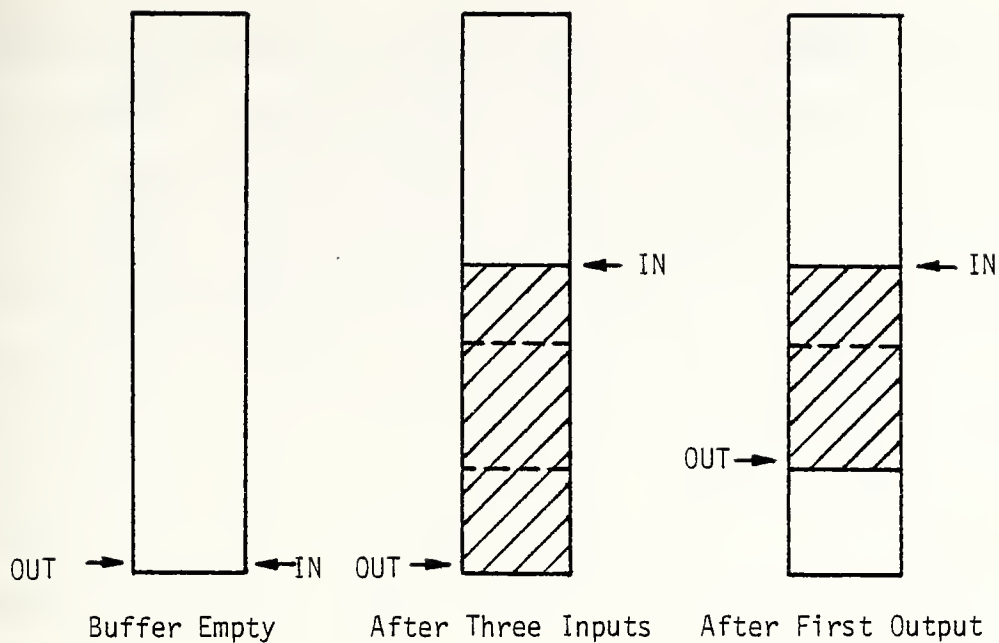


Figure 9: 8748 Buffer Operation

computed by using the current values of the input and output pointers. When the input pointer is greater than the output pointer, the available room is at the top and bottom of the buffer. The available room is the size of the buffer minus the difference between the input and output pointers. When the output pointer is above the input pointer, the available room is in the middle of the buffer and is equal to the output pointer minus the input pointer. An output operation is allowed to begin in the middle of an input operation since the output operation can only increase the amount of room in the buffer. Likewise an input can begin during an output operation, since the available room at the beginning of the input operation can only increase with time. To accommodate long messages, the input lock is loaded with the requesting computer number at the time of the request even though space may not be available at that time. All other message inputs are rejected until the computer in the input lock can transfer its message. This prevents a number of short messages from pre-empting a longer message.

F. SERIAL INTERFACE TERMINAL OPERATION

The serial interface terminal was viewed as a synchronous machine with several distinct terminal states corresponding with different stages of message handling.

The terminal state is defined by a four bit quantity stored in a working register of the 8748 computer chip.

The states are divided into two groups; the listening states - 0000 through 0111, and the talking states - 1000 through 1111. Listeners change state by receipt of commands from the talker, the bus current bus control station. The talker changes state after receiving an acknowledgement of the last command. While in the talker, or bus control states, any message received after the issue of a command is considered an acknowledgement - no parity or ID checks are made. While in the listener mode, messages are scrutinized for proper ID numbers and parity to prevent terminals from responding to the wrong commands. State transitions are diagrammed in figure 10.

State 0000

This is the 'dormant' state of the terminal while it is monitoring the bus. The terminal is not engaged in any serial transfer. All received messages are examined for possible commands addressed to the particular terminal.

State 0001

A terminal enters this state after sending a clear-to-send acknowledgement in response to a request-to-send command from the current bus controller. The acknowledgement is sent only if space is available in the buffer to accommodate the message. The terminal will stay in state

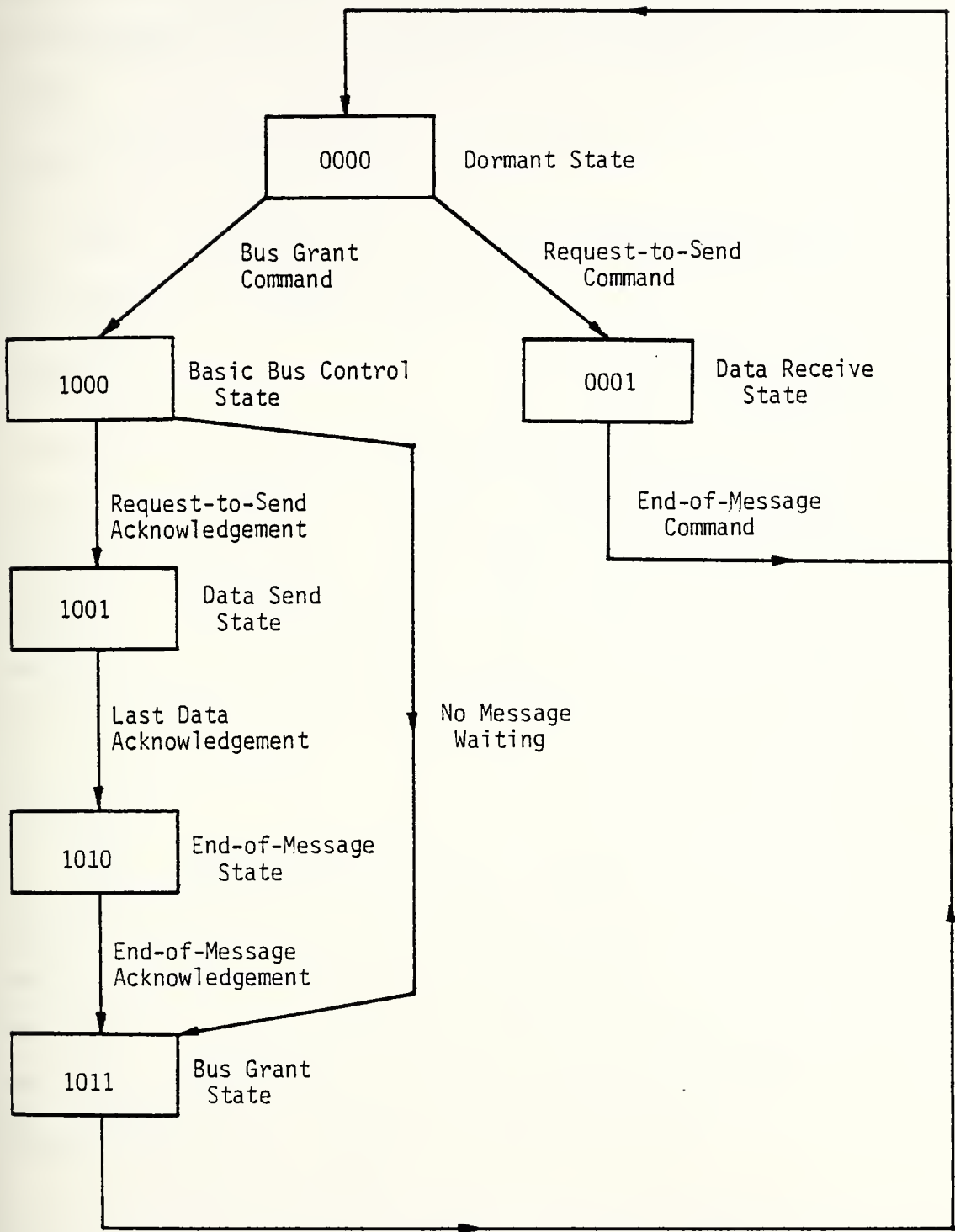


Figure 10: Serial Interface Terminal States

0001 during the reception of the message, sending an acknowledgement after the successful decoding of each data word.

States 0010 through 0111

These states are not used in the present design and are reserved for future expansion.

State 1000

This is the basic bus control state of the terminal. While in this state, the terminal checks for any messages awaiting transmission. If a message is waiting, the bus controller sends a request-to-send command to the desired terminal. If no traffic is pending, the terminal immediately enters state 1011.

State 1001

This state is entered after receiving a clear-to-send acknowledgement from the terminal addressed while in mode 1000. The terminal sends the message while in state 1010 and remains in this state until all the data words have been sent. The next data word is issued only after an acknowledgement is received for the last data.

State 1010

After receiving an acknowledgement for the last data word in the message, the bus controller enters state 1010. During this state, the end-of-message command is issued.

State 1011

State 1011 is the bus grant state and is reached by either of two ways: by lacking any traffic to send while in state 1000, or by receiving an end-of-message acknowledgement while in state 1010. The bus controller relinquishes the bus by issuing a bus grant command to the next terminal to receive control. The terminal considers the next bus transmission a bus grant acknowledgement and switches back to state 0000.

States 1100 through 1111

These states are not utilized in the present design and are reserved for future use.

IV. HARDWARE DESIGN

The serial interface hardware is partitioned into three major sections, as shown in figure 11. These are the serial bus interface, the parallel bus interface, and the CPU group. Each of these sections will now be discussed in greater detail.

A. THE SERIAL BUS INTERFACE

The serial bus interface comprises the major portion of the hardware and is divided into five subsections: the transmitter, receiver, clock synchronizer, sync detector and the controller.

1. Transmitter

The transmitter is composed of three functional blocks, the transmitter parallel-to-serial shift register, the transmitter parity generator, and the encoder. The parallel-to-serial shift register is 16 bits in length, made up of two eight bit shift registers tied end to end. Provisions are made for expansion to 32 bits with the addition of two more IC chips. The shift registers are loaded individually by the CPU by applying the desired eight bits of data on the CPU data bus and then pulling the desired load line low. During transmission, the shift registers are clocked 16 times, presenting serial data to the encoder. The output of the last shift

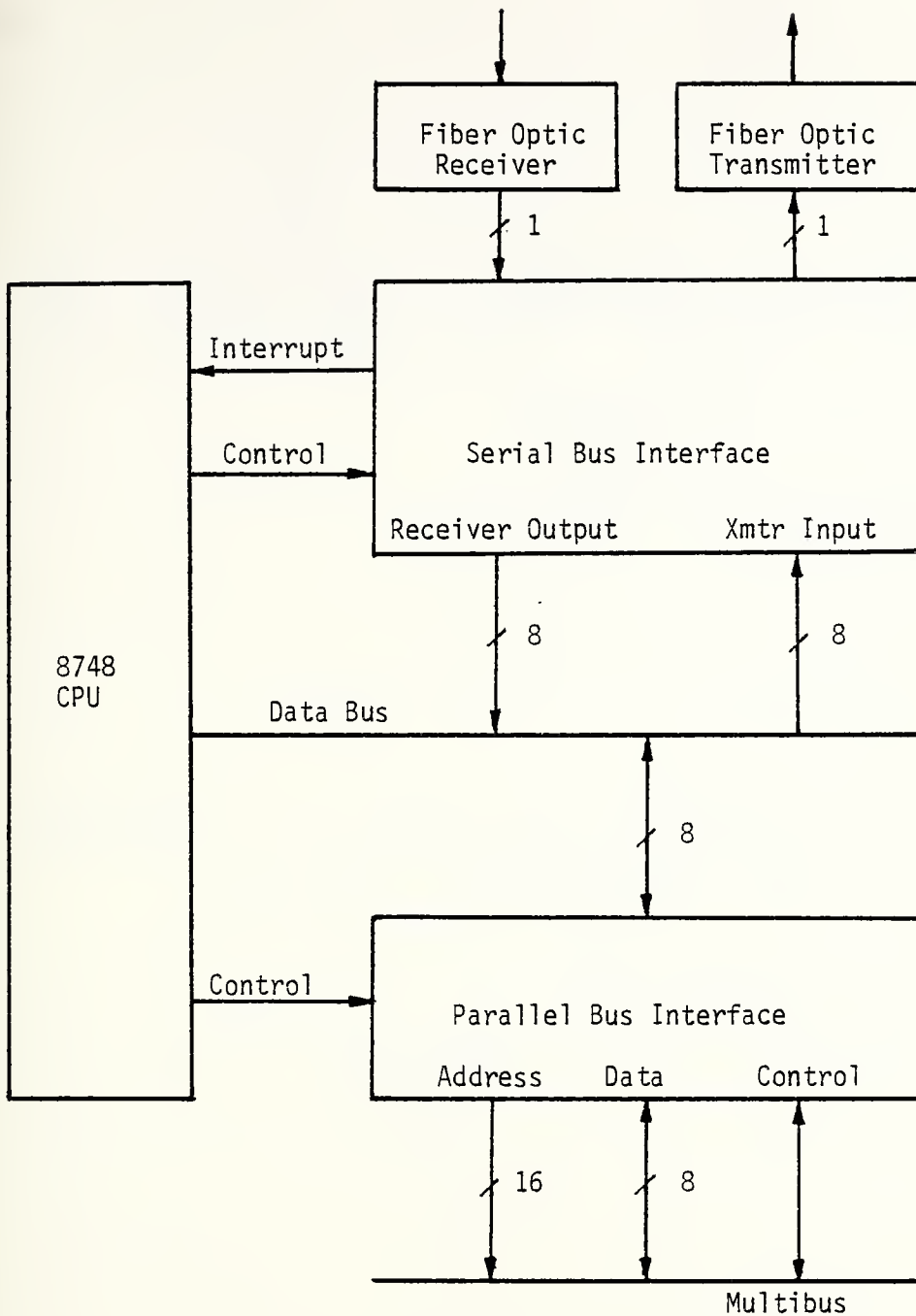
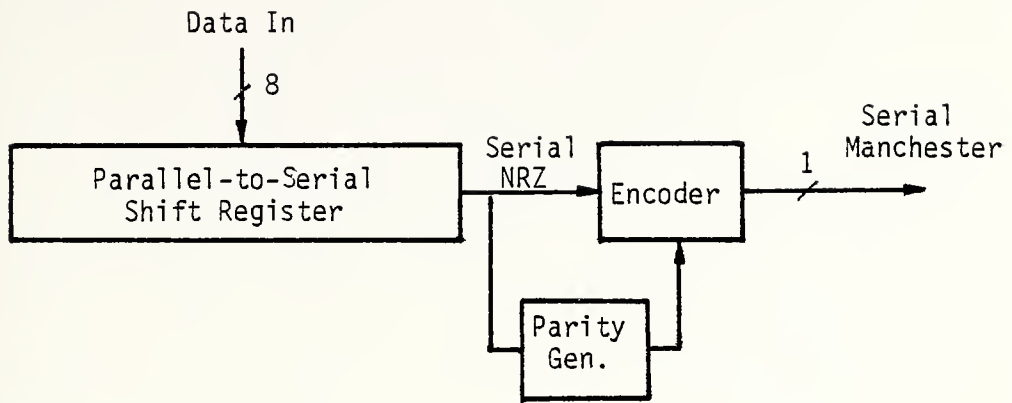
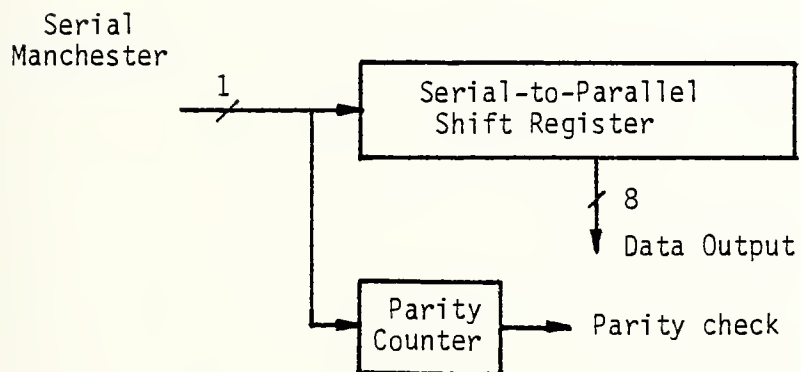


Figure 11: Serial Interface Block Diagram



Transmitter



Receiver

Figure 12: Transmitter and Receiver Block Diagrams

register, the one closest to the encoder, is fed back into the input of the first shift register to provide an automatic reload of the data. Reloading the shift registers is not required to repeat a message.

The transmitter parity generator consists of a counter to accumulate the number of logical ones in the outgoing data. The counter is activated by the same clock which shifts the data and stops counting after the sixteenth bit has been sent. At this time the least significant bit of the counter tells whether the number of ones in the message is even or odd, by being a 0 or a 1 respectively. Parity is taken over all 17 bits, message plus parity. If even parity is desired, the counter output is taken directly as the parity bit. If odd parity is desired, the least significant bit of the counter is inverted. For example, if odd parity is desired, and the number of ones is even, the zero level of the counter LSB is inverted making the parity bit a one and making the number of ones in the entire 17 bits odd. Parity sense is selected by the use of an exclusive-or gate on the output of the parity counter.

The encoder converts the NRZ binary data out of the shift registers into a unipolar Manchester code for transmission. The encoder consists of a D-type flip flop and several gates acting as a multiplexer. The encoder generates the Manchester code by sending the

data during the first half of the bit time and sending the complement of the data during the second half. Additional control signals allow the generation of the non-Manchester sync characters and allow insertion of the parity bit following the last data bit. These control signals are derived from the controller.

2. Receiver

The receiver consists of a parity counter and a 16 bit serial-to-parallel shift register expandable to 32 bits. Due to the word width limitation of the controller ROM, the shift register and parity counter operate off the same clock. Since the parity check is performed over the entire 17 bits, 17 clocks are required by the parity counter. To allow the receiver shift register to function properly, the receiver data is delayed one bit time with respect to the data into the parity counter. At the end of the receive sequence, a flip flop is set to signal the CPU that a message has arrived. The parity counter functions identically to its counterpart in the transmitter. The results of the parity count are inverted, if required, by an exclusive-or gate and passed on to the CPU.

3. Clock Synchronizer

The clock synchronizer keeps the local clock synchronized with the transmitting terminal's clock during the reception of a message. The clock sync functions

by detecting the transitions in the received Manchester data. The clocks for both transmission and reception are derived from a high frequency clock of approximately 10 MHz. During transmission, the basic clock frequency is divided by six to provide a nominal 768KHz data rate. During receive, this basic frequency is divided by seven to provide a slightly lower frequency for the clock sync.

The clock sync, figure 13, consists of a divide-by-seven counter and a transition detector. The counter is presettable and can be reloaded by either an overflow or a signal from the transition detector. Each time a transition is detected in the received data, the divide-by-seven counter is reset. During a received string of ones or zeros, transitions occur both at the beginning and in the middle of the bit times. The divide-by-seven counter will be reset by the incoming transitions every six 10MHz clocks, and will never reach terminal count. During alternating one-zero strings, transition occurs only in the middle of the bit intervals, or once every twelve 10MHz clocks. Under these conditions, the divide-by-seven counter will reach terminal count, reset itself and count five more times before being reset by the next transition. The clock sync thus 'free runs' during the times when transitions do not occur. During the reception of a sync character, the divide-by-seven counter may overflow up to three times before being reset by the data. The ratios

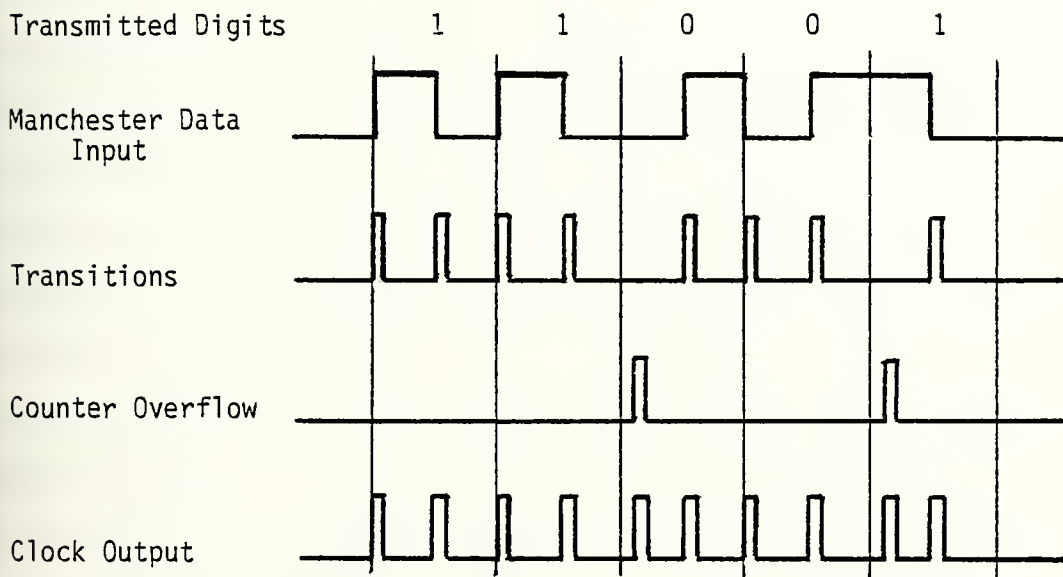
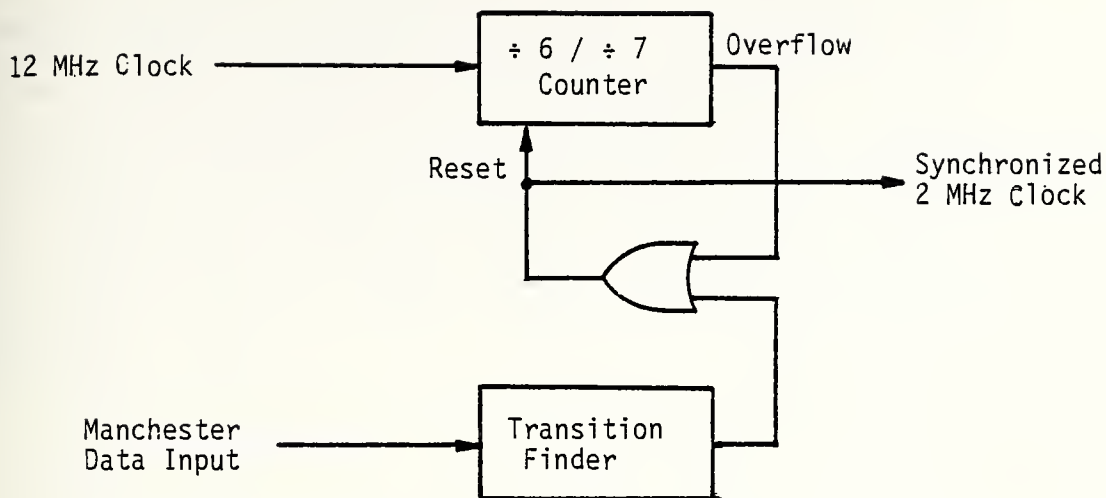


Figure 13: Clock Synchronizer Operation

of six and seven were chosen to keep the receive clock always behind the sending clock and to not lose a clock pulse during reception of the sync characters. During transmit, the clock synchronizer is disabled to prevent the received echo from altering the transmitter clock. The clock divider is forced into a divide-by-six mode during this time.

4. Sync Detector

The sync detector is a digital matched filter tuned to the sync characters used in the system. The matched filter responds to three different bit patterns to accommodate both the data sync and command sync characters shown in figure 14. For data sync the filter responds to the bit pattern 111000. For command sync, two patterns, 011101 and 011110, are detected to handle the cases of command sync-leading zero and command sync-leading one. Both patterns must be detected to distinguish command sync from data sync. The output of the sync detector is a pulse which starts the controller to generate the signals needed to receive the message. Also included in the sync detector is the circuit which starts the controller during transmit. During transmission, the matched filter is disabled to prevent the received echo from re-loading the controller.

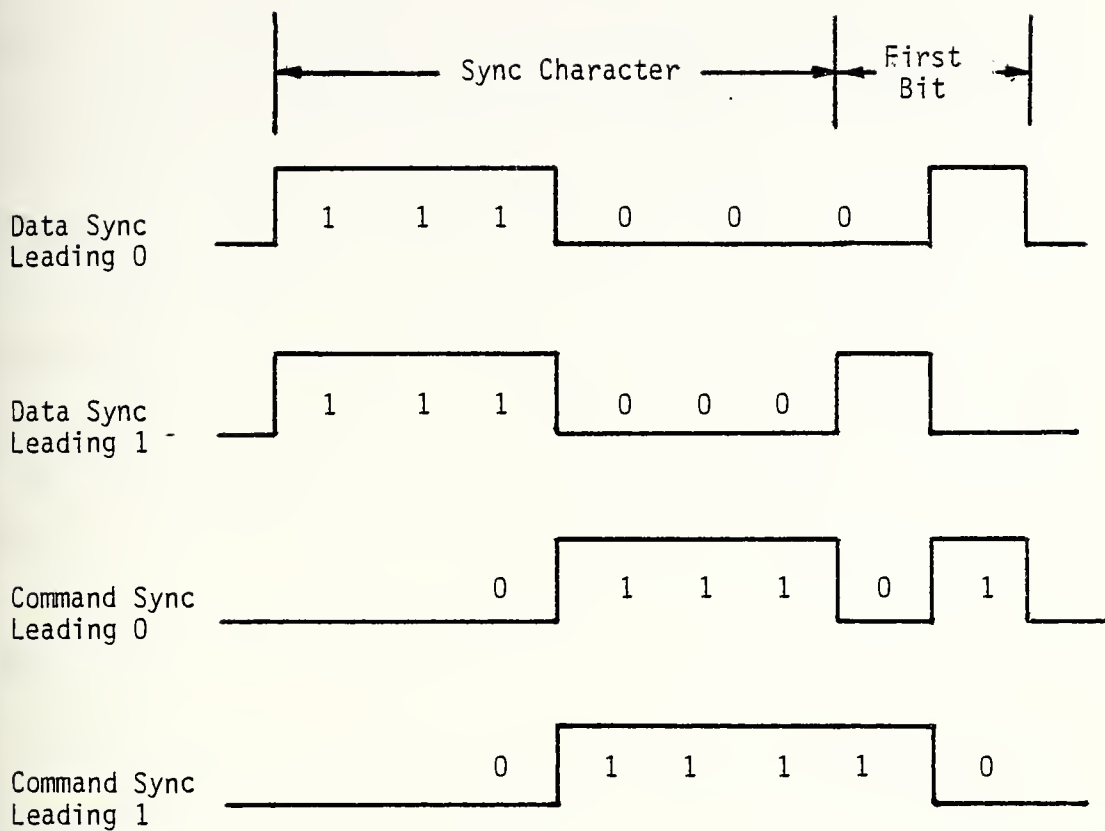


Figure 14: Sync Waveforms

5. Controller

The controller is programmed via an EPROM to provide all the control signals needed by the transmitter and receiver. It consists of an eight bit counter and a 1024 word by eight bit memory. The memory outputs are clocked through a register to eliminate any glitches on the resulting control lines. The 1K by eight EPROM can accommodate four sets of transmit/receive programs for 16 bit data, or two sets of programs for a 32bit/message system. Provisions are made for easy change-over to a 2048 by eight EPROM which will double the above figures.

B. PARALLEL BUS INTERFACE

The parallel bus interface provides the signals needed to communicate with peripherals via the Intel Multibus. The circuitry provided in the serial interface unit has limited capabilities in that it can communicate only with external memory. No local bus clock is provided since the bus clock is provided by one of the single board computers residing on the parallel bus. The parallel exchange is initiated by the CPU on the serial interface board. A bus request signal is issued in sync with the bus clock. Before the next clock pulse, a bus arbitration decision is reached, and the module to receive bus control is issued a bus priority input signal. The module waits until the current user of the bus has released the BUSY line before taking control of the bus.

Address and data are applied to the bus by activating tri-state drivers. The appropriate read or write command is issued after the data and address have had time to settle and removed after the acknowledge has been received from the external device. The bus is relinquished after the transaction by turning off the data and address drivers and releasing the BUSY line.

C. CPU GROUP

The Intel 8748 is the central processor unit of the serial interface board. The 8748 is a single chip computer with 64 bytes of read-write memory and 1K bytes of electrically programmable/erasable read-only memory. The 8748 provides 27 input output lines grouped into two eight bit ports, one eight bit bus, two test inputs, and one interrupt input. The CPU chip controls all the activity on the serial interface boards with the exception of the automatic functions in the receiver. The 8748 loads data into the transmitter, initializes serial transmissions, decodes received messages, and controls the parallel interface to corporate memory. External latches are used to expand the number of output lines from the 8748.

D. ELECTRO-OPTIC COMPONENTS

This project used commercially available electro-optic and optical components. Each serial interface

board requires a fiber optic transmitter and receiver to connect to the serial bus. The serial bus is a star arrangement with separate lines from each transmitter and receiver connected to a single coupler, as shown in figure 15. The optical transmitter and receiver are housed in separate enclosures and provide TTL level inputs and outputs. The transmitter, receivers, and radial coupler were manufactured by Spectronics, Inc. of Richardson, TX. The fiber optic cable was made by Galileo Electro-Optics.

E. CONSTRUCTION DETAILS

The serial interface hardware is built on an Intel Multibus compatible 12 by 6.75 inch circuit board. The electrical interconnections are wirewrapped to facilitate modification and expansion. To reduce power consumption, the serial interface hardware utilizes low power Schottky TTL devices in the majority of the circuitry. The 8748 computer chip is a five volt NMOS circuit while the controller EPROM requires additional +12 and -5 volt power. All connections to the parallel Multibus are made through an edge connector on the circuit board. Connections to the fiber optic transmitter and receiver are made with coaxial cables directly to the board.

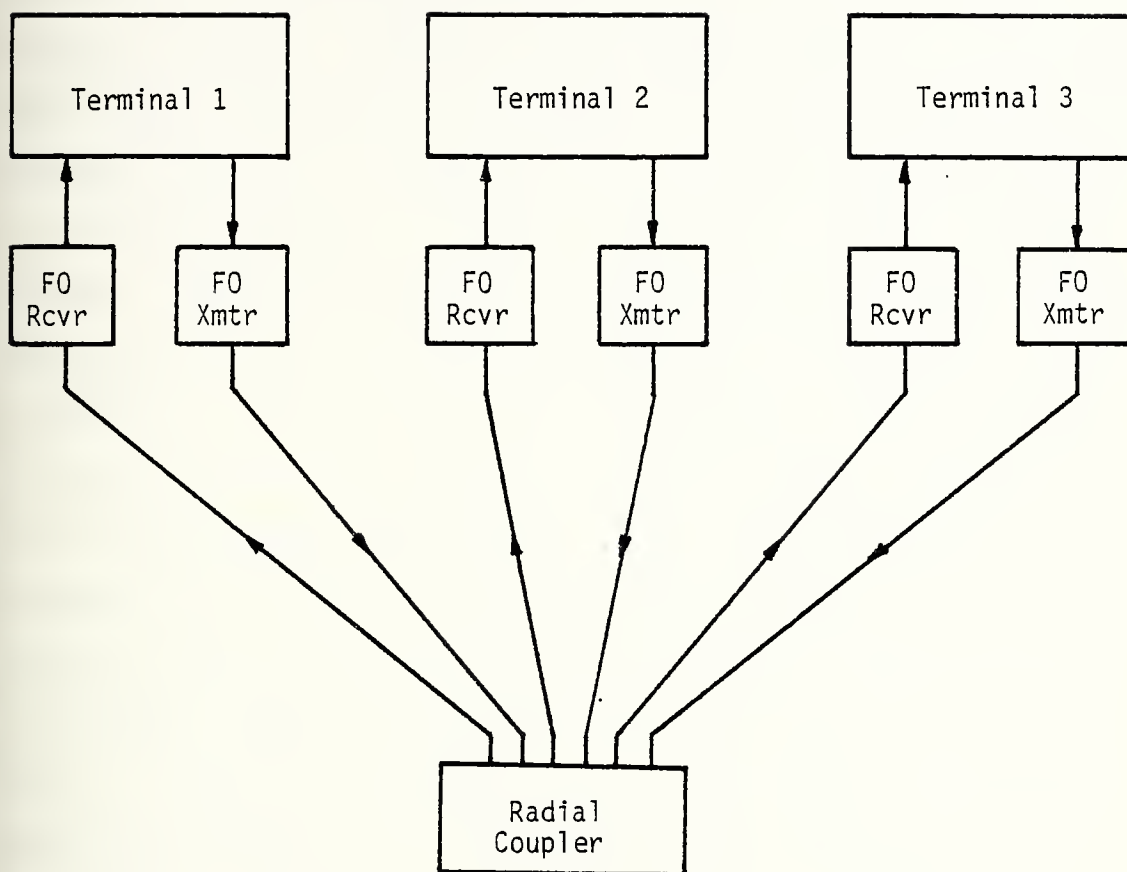


Figure 15: Optical Data Bus Configuration

V. SOFTWARE DESIGN

The serial interface software is composed of an executive program, parallel and serial message handling routines, a time out interrupt handler, and a power-up restart routine. The executive program (EXEC) is a continuous loop checking the 8748 buffer area and the corporate memory buffer for messages and calling the service routines to transfer messages to and from the 8748 buffer. This tends to keep the buffer as empty as possible. EXEC checks the output lock of the 8748 buffer, if it is locked it signifies that a serial output is in progress. If the output lock is not set, EXEC reads the receiving module number of the next outgoing message and places the corresponding SBC number in the lock. If the message is bound for a remote location, EXEC sets an internal flag in the 8748 to signal the serial message handler that a message is waiting to be sent. If the next message is local, EXEC calls the parallel message delivery routine. When no more messages can be removed from the 8748 buffer, EXEC checks for incoming messages in the corporate memory message buffer.

A. PARALLEL MESSAGE HANDLING ROUTINES

Parallel messages are received and delivered by five routines called by EXEC. Two minor routines, SETLOK and UNLOCK, lock and unlock the corporate message buffer. RECEXT receives a message from the corporate buffer.

This routine is called by EXEC after it has determined that the next outgoing message in the corporate buffer should be taken into the serial interface. RECEXT checks the available room in the 8748 buffer before commencing the data transfer. If no room is available at a given time, the process is aborted. SENDEXT is the message delivery counterpart to RECEXT. SENDEXT checks the available room in the corporate buffer before sending a message, and like RECEXT, aborts the process if insufficient space exists. One additional routine, SETEXT, sets the EXTMSG register in corporate memory signaling the next SBC to receive an outgoing message.

B. SERIAL MESSAGE HANDLING ROUTINES

All serial message handling is interrupt driven. Received messages cause an interrupt to the message decoder routine MSGDEC. This routine examines the destination address of the message to determine if it is addressed to that particular terminal. MSGDEC also checks the parity and type of message based on the sync character. MSGDEC calls the other serial message handling routines based on its findings. If the terminal is in a receiving mode, the routine RECMSG is called to handle the data byte, storing it in the 8748 buffer. If the terminal is in a listening mode, the command handlers, BUSCMD, REQCMD, and ENDCMD handle the bus grant command, the request-to-send command, and the end-of-message command. If the terminal is in a bus control mode, the

received message, regardless of its type, is considered as an acknowledgement, and is processed by one of the acknowledgement handlers, BUSACK, CLRACK, DATAACK, or ENDACK. These handle the bus grant, the clear-to-send, the data message, and the end-of-message acknowledgements respectively. At the conclusion of the interrupt sequence, control is returned to EXEC.

A separate routine, TIMOUT, handles the time out interrupts generated by the internal timer/counter. The timer is reset each time a message is received and each time a message is sent. If the terminal is in a bus control mode, a time out will initiate a repeat of the last message sent. Terminals which are in a listening mode may also experience time out interrupts but do not take any action. After a set number of consecutive time outs without a received message, the terminal will restart its initiation procedures, thus aborting all undelivered messages. The reset operation will allow the terminals to rapidly resume normal activity once the fault has been removed.

The RESTRT routine initializes the terminal after power-up and time out generated resets. RESTRT reads the locations in corporate memory for the necessary identity information and local bus roster, clears the internal registers of the 8748 and initializes the pointers for the 8748 buffer memory. If the terminal is in the initial bus control terminal, RESTRT initiates the first bus grant command. After

completing the restart procedure, the program control is turned over to EXEC. Details in the individual routines can be found in Appendix B.

VI. SYSTEM INTEGRATION

The system hardware was initially breadboarded and debugged before the wire-wrap boards were constructed. Software check-out was delayed until a working board could be interfaced with the Multibus. The operating software was written by hand in 8748 assembly language. To facilitate debugging, the hardware was modified to include 1024 bytes of on-board read-write memory for program storage. The program was loaded into memory by an interconnecting cable to an SBC 80/20 computer. The 8748 was operated in an external instruction fetch mode and programs could be modified without erasing and reprogramming the internal EPROM. Using the same 80/20 interface, the 8748 was single stepped through the program and addresses were displayed on a CRT terminal. Contents of the internal registers could be examined only by modifying the 8748 program to dump their contents into an external memory board on the Multibus. The external memory could then be read by the 80/20 monitor.

The SBC 80/20 used for debugging was resident in an Intel MDS microcomputer system. Programs were stored in MDS memory and transferred to floppy disk at the end of each debugging session. Another SBC 80/20 in a separate card cage was used to enable two serial interface boards to be connected and debugged simultaneously. Messages were simulated by loading data into the proper locations of the

external (corporate) memory. Additional details on debugging are included in Appendix C.

VII. CONCLUSIONS

After construction and debugging, the hardware was tested and was found to function precisely as designed. The software took considerably more effort to debug and some minor difficulties exist in the parallel message handling routines presented in this thesis. Although the software was not fully operational, the serial message handling and bus control techniques were verified and were found to function as desired.

The software used in the serial interface is determined by the message handling techniques utilized by the distributed operating system. As the operating system is changed the serial interface programs must be modified.

This project should be viewed as a working test bed to evaluate message handling techniques and distributed operating systems. Fiber optic components used in the data bus can also be evaluated under actual operating conditions.

The serial message handling and bus control techniques used in this system were chosen partially for their ease of implementation and deserve further study to develop more efficient routines. The major software limitations were imposed by the modest program memory of the 8748 computer chip. Future designs could employ the Intel 8049 chip to provide twice the program and data memory along with increased operating speed. Different message lengths could be

used to allow more efficient transfer of data. The software could allow for changing message formats and the entire bus control procedure while the system is in operation. This could be used to advantage in tactical data systems where a different operating mode would be required in a battle situation.

APPENDIX A

HARDWARE REFERENCE MANUAL

This section describes in detail the circuit operation of the serial interface hardware.

A. CLOCK SYNC

1. Theory of Operation

The clock sync circuit keeps the receiver clock in step with the transmitting clock for proper decoding of the received data. Manchester coded data always undergoes a transition in the middle of each bit interval. A logical one would be high for the first half and low for the second half. The reverse is true for a logical zero. Decoding the Manchester signal is accomplished by sampling the incoming data during the first half bit time.

The basic clock in this system is 12 times the transmitted bit rate, providing six clocks per half bit time. For convenience, the basic clock will be referred to as the 12 MHz clock; normalizing the data rate to 1 Mbit/sec. With six 12 MHz clocks in the half bit time, the best place to sample the data is on the rising edge of the third clock into the interval. However, if the starting point of the interval can be determined, sampling any time during the first four clocks is satisfactory.

This circuit relies on the transitions of the received data to keep the local clock in sync. Depending upon the data, transitions occur at least once, and as often as twice, per bit time. These transitions are used to reset a divide-by-seven counter. The counter free runs during times when no transitions are detected. If transitions occur both at the beginning and at the middle of each bit time, the counter will be reset every six clocks and never reach its terminal count. If transitions occur only once per bit time, the counter is reset every 12 clocks, overflowing after seven clocks and being reset during the fifth count after overflow.

Forcing the counter to divide by seven insures that the receiver clock will always lag behind the transmitting clock. In this way the receiver can never generate an extra clock. During the sync characters, the divide-by-seven counter may overflow three times before being reset by a transition. The worst case is a command sync character followed by a one in the data. Here the received signal is high for two full bit times or twenty-four 12 MHz clocks. The divide-by-seven counter is reset at the beginning and overflows three times after 21 clocks. The counter is reset in the middle of the first data bit after counting three times since the third overflow. The resultant clocks are spaced at irregular intervals: 7, 7, 7, and 3, but the decoding is not affected and the clock remains in step with the transmitter.

2. Circuit Description

Transitions are detected by a transition finder composed of a 74LS175 flip flop, B9, and a 74S00 NAND gate, B10. The incoming data, FDATA, is shifted through the first two stages of B9 and the NAND gates form an exclusive-or gate to detect the transitions. The output of this circuit is an active high pulse lasting one 12 MHz clock and occurring immediately after the rising or falling edge of the data. The output of the second stage of B9, CDATA, is the data input to the sync detector. Figure 16 shows the clock sync schematic.

The 12 MHz clock is divided by a 74LS161 presettable counter B12. This counter can be preset to any value and activates a terminal count (TC) output upon reaching a value of 15. The counter division ratio is controlled by the transmit mode control, loading the counter with ten to divide by six during transmit, and loading nine to divide by seven during receive. The counter is reset by either the terminal count output or by the output pulse of the transition finder. The output of the clock sync is a 2 MHz clock which is derived from the logical or'ing of the two counter load pulses. The NAND gates of B11 perform this operation in addition to locking out the transition pulse during transmit. The output of the clock sync is reclocked with the 12 MHz clock to prevent clock skew from gate delays. Due to the number of gate delays in the counter reset path, gates B10 and B11

are Schottky TTL to allow reliable operation with the low power counter and flip flop at the operating frequency.

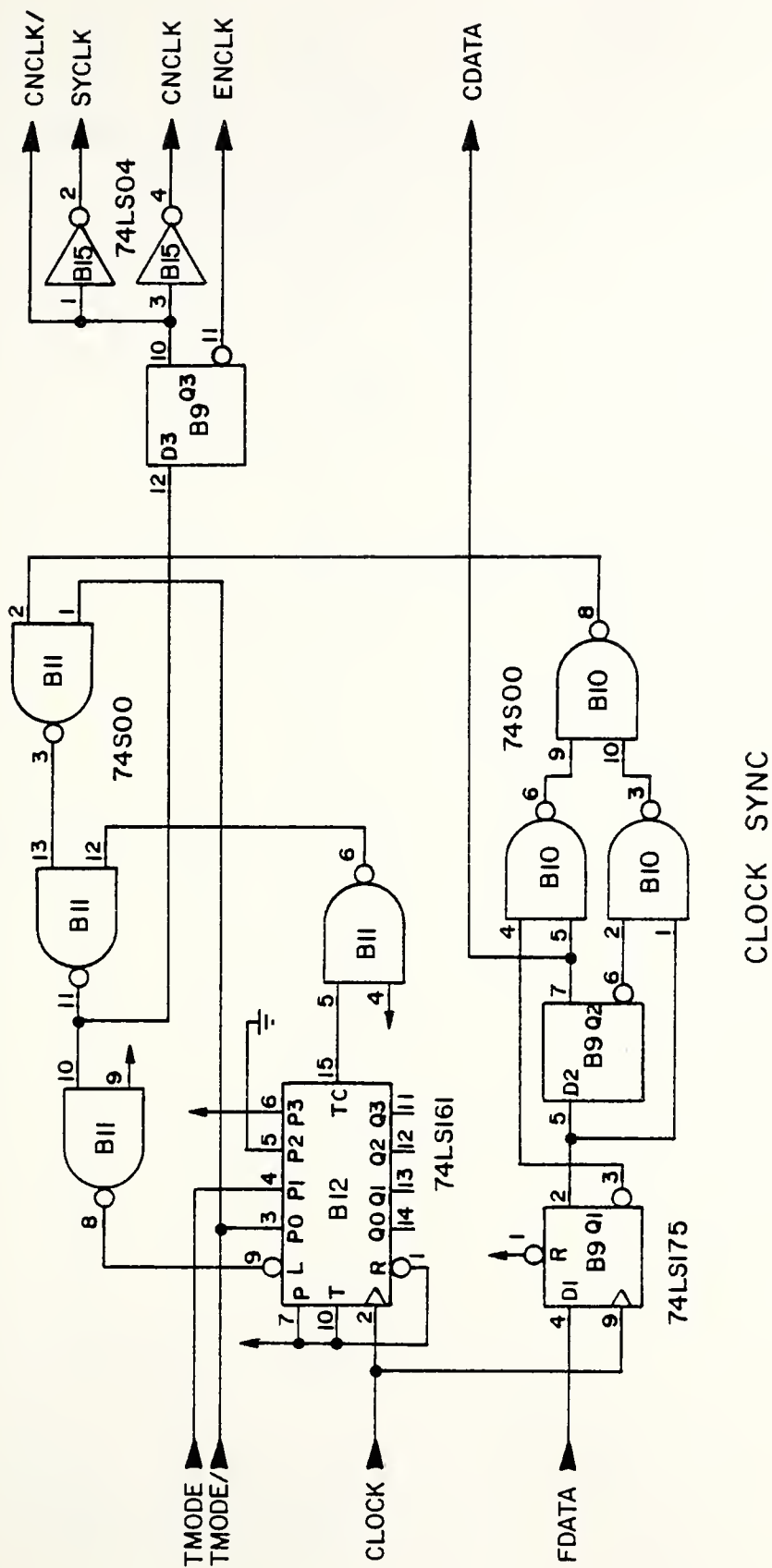


Figure 16: Clock Sync Schematic

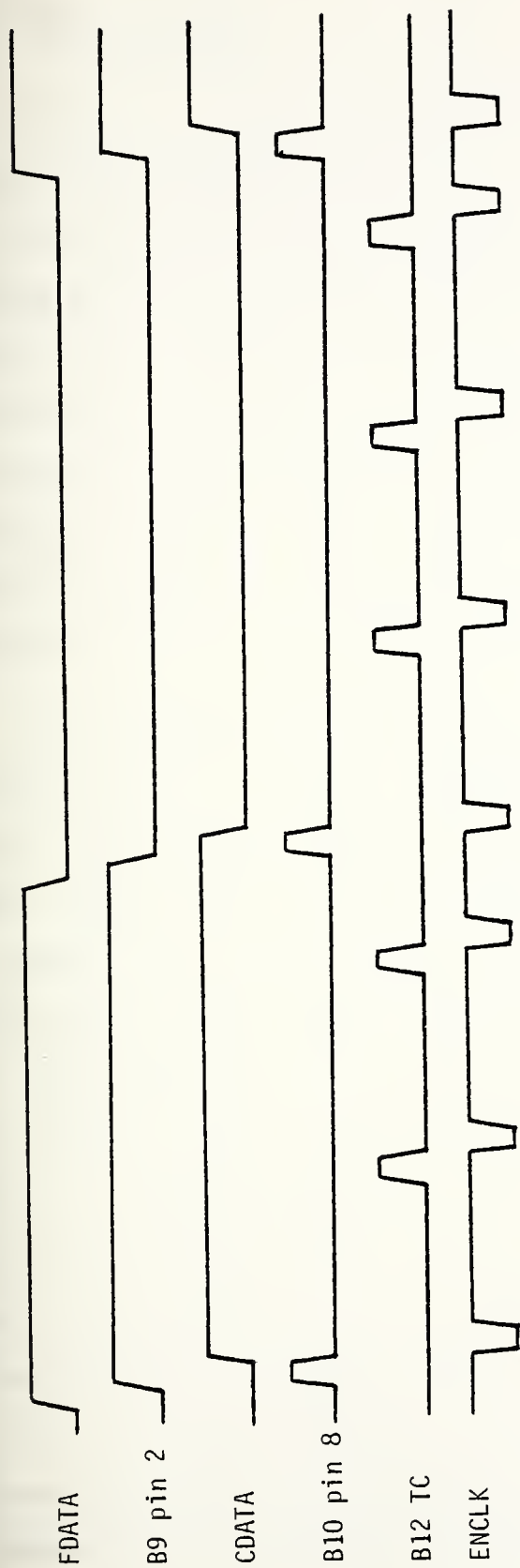


Figure 17: Clock Sync Timing

B. SYNC DETECTOR

The sync detector determines the precise instant at which the decoding process is started. This circuit utilizes a matched filter composed of an eight bit shift register and three eight input NAND gates. The inputs of the gates are connected to the appropriate outputs of the shift register to respond to the desired bit pattern. The shift register is clocked by the 2 MHz clock from the clock sync, SYCLK. Different sync characters for data and command messages are used in this system. The first half of the command sync is not transmitted.

Detecting the sync characters reduces to recognizing the pattern in the shift register. A data sync is detected by a high level for three half bit times followed by a low for three half bit times, corresponding to a pattern of 111000 in the shift register. Command sync detection involves the sense of the first data bit in the message. If the leading bit is one, the command sync is recognized by a high level for four half bit times, or a pattern of 1111. If the first bit is a zero, the pattern becomes 1110. To separate this last case from the data sync, the next half bit time is also detected, making the command sync patterns 11110 and 11101. A leading zero is added to the two patterns above to keep the 11110 pattern from being decoded as a 11101 after one shift. This makes the command sync patterns 011110 and 011101.

Since the data sync begins before the command sync, the data sync gate is shifted down the register by two taps. The data going into the receiver is tapped from the shift register adding delay to the data to compensate for the delay involved in the sync detector output. The sync detector schematic is shown in figure 18.

Flip flops A9 and A10 are 74LS175 flip flops wired as shift registers. The decoding gates are 74LS30's A11, A12, and A13. The outputs of the eight input gates are or'ed in a four input 74LS20 to form a single load pulse output to the controller. The output of the data sync gate A13 is inverted and passed on to the controller to load the proper program for decoding the particular sync type. Gate B14 performs this inversion in addition to allowing the CPU to select the desired sync pattern during transmit.

The signal out of gate A14, PGCLR/, is reclocked to remove any gating glitches and is used to reset the parity counters in both the transmitter and receiver. The load pulse to the controller, CLOAD/, is not delayed in order that the data sync/command sync information, SYMOD, from gate B14 is valid at the time the controller is loaded. Glitches on CLOAD/ are not a problem since the load operation is synchronous with the 2 MHz clock.

Flip flop B13, a 74LS74, detects the falling edge of the CPU send request, SSREQ/, and generates a pulse to load the controller during transmit. Flip flop A15 is set when en-

tering the transmit mode and cleared when the transmission is completed. The output of this flip flop, TMODE, locks the sync detector during transmit to prevent the received echo from reloading the controller. An additional lockout input, MSGRY/, comes from the CPU which prevents a message reception until the last message has been serviced.

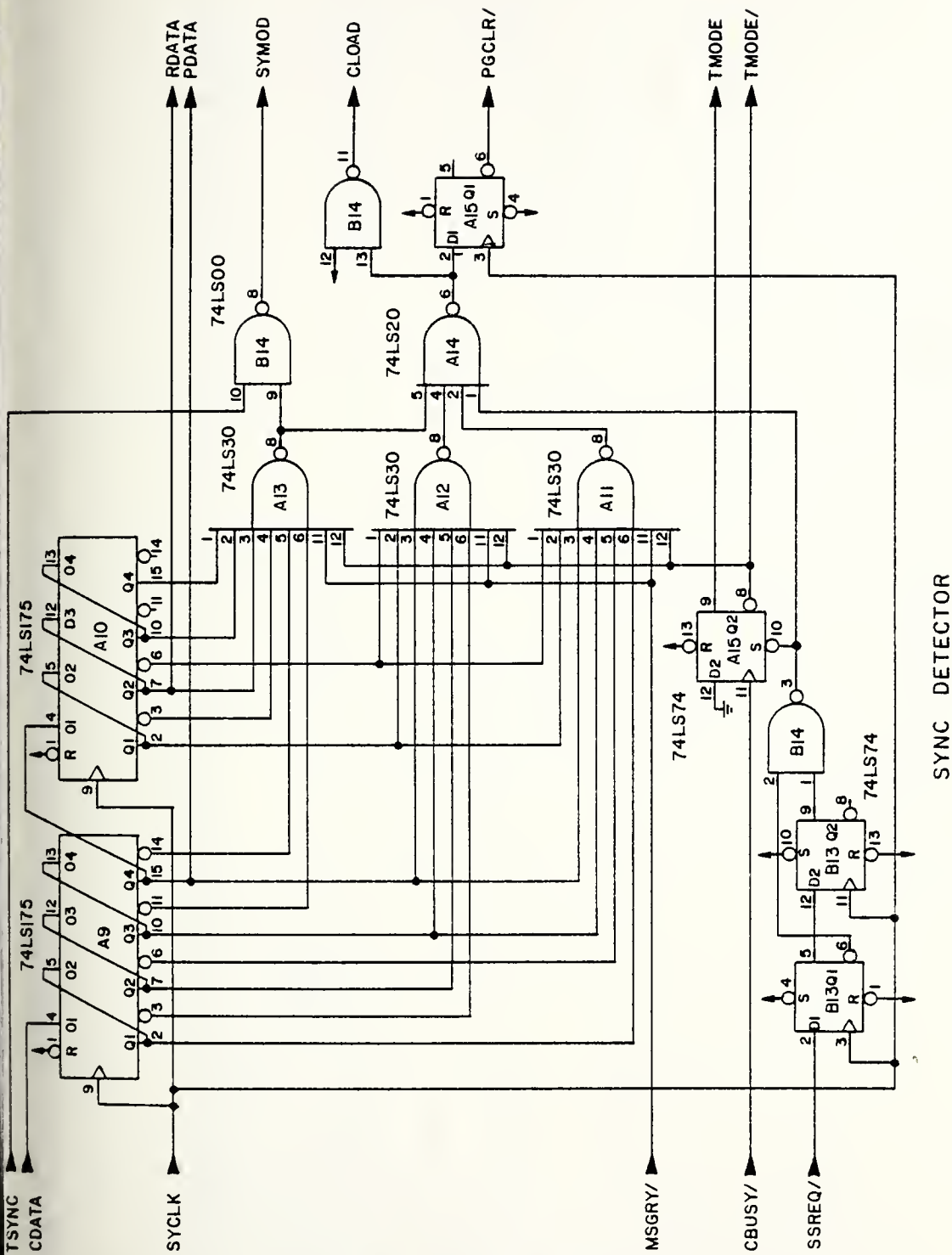


Figure 18: Sync Detector Schematic

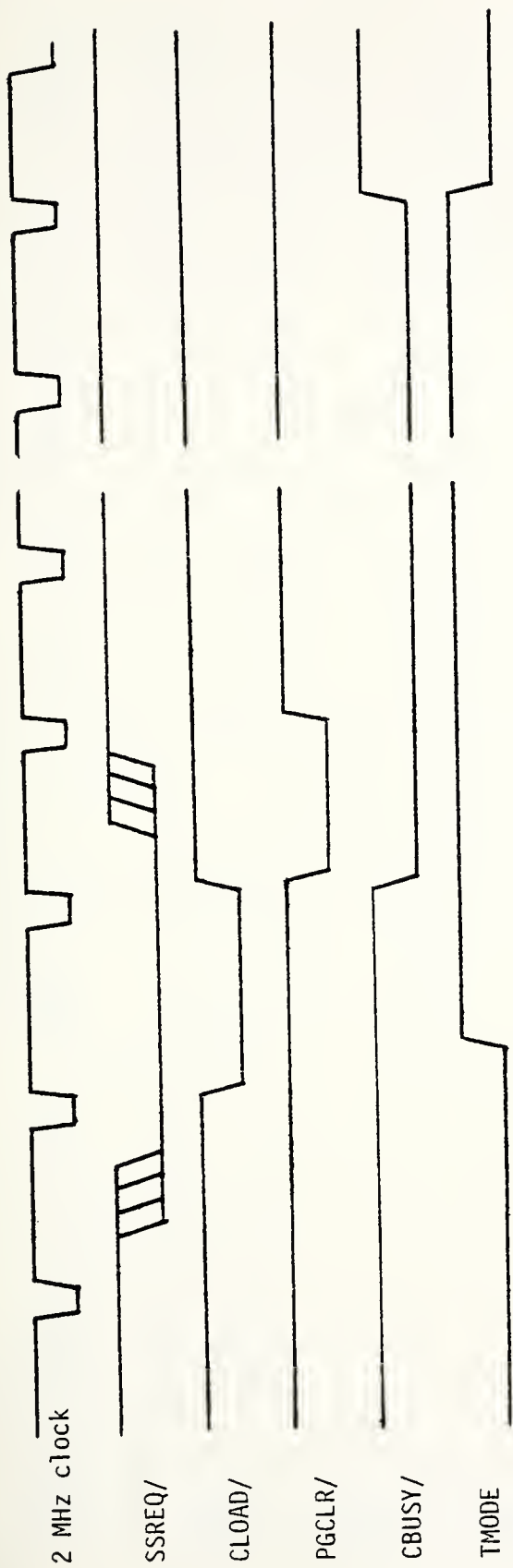


Figure 19: Sync Detector Transmit Timing

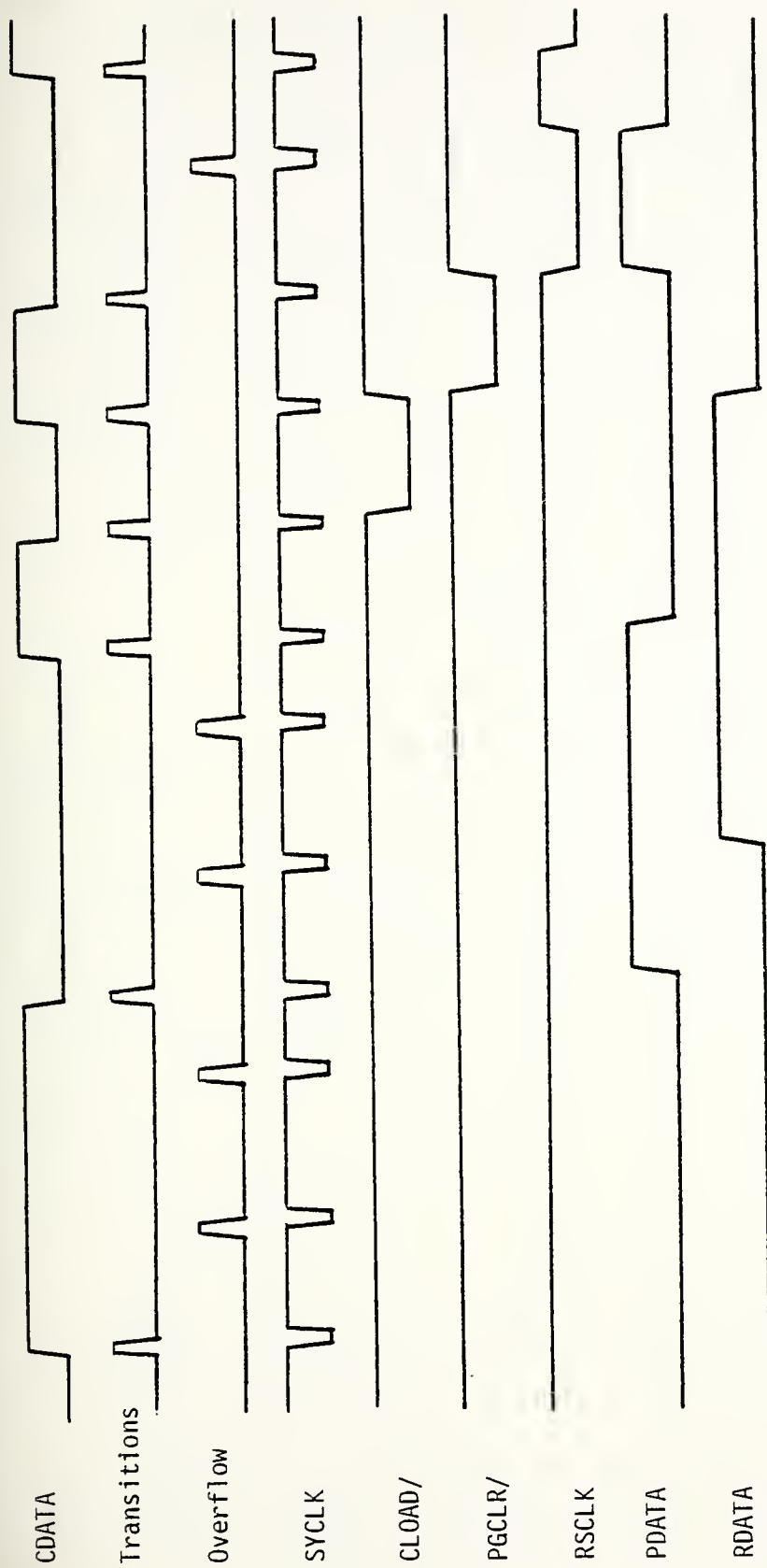


Figure 20: Sync Detector Receive Timing - Data Sync

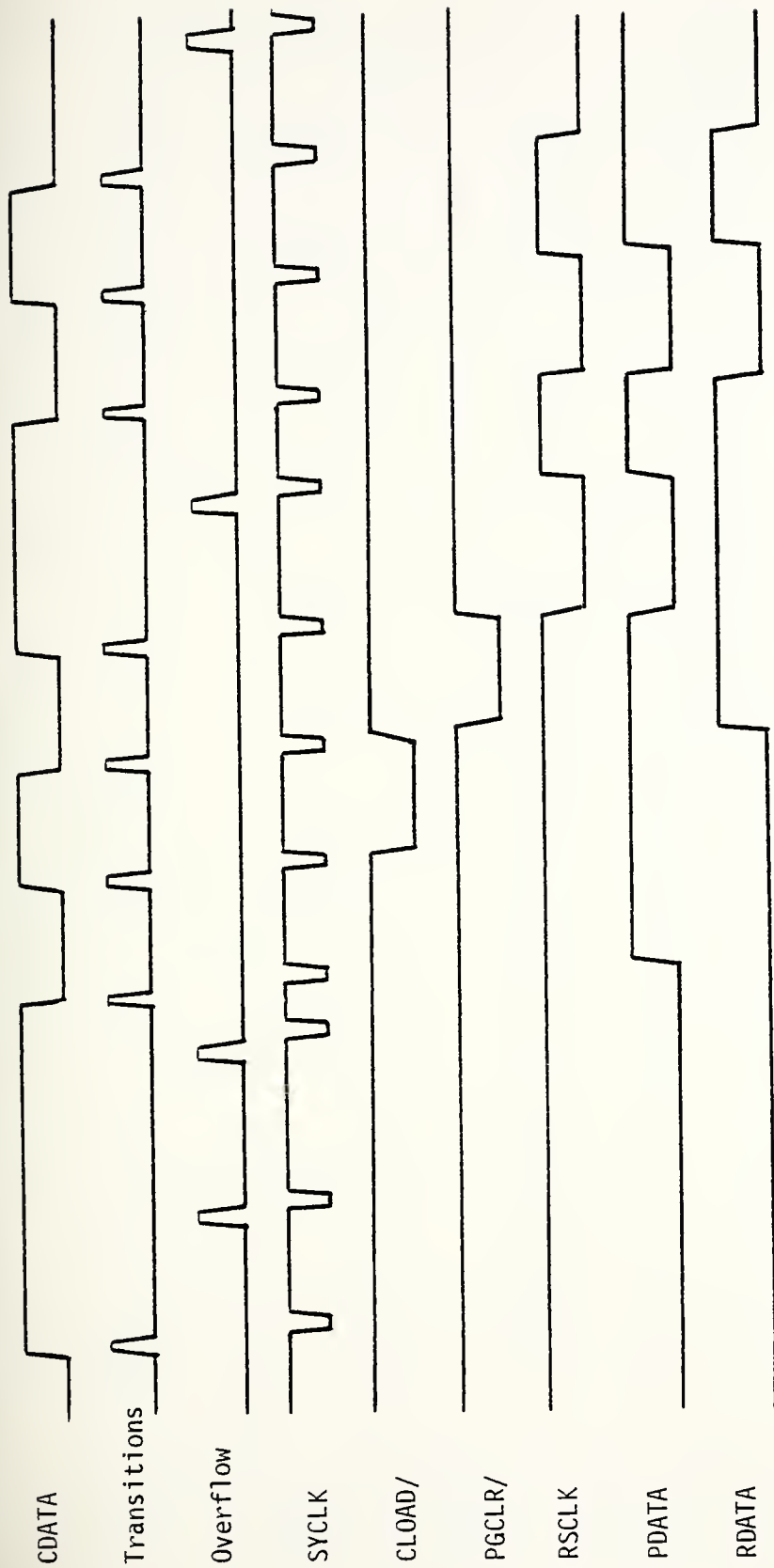


Figure 21: Sync Detector Receive Timing - Command Sync, Leading 0

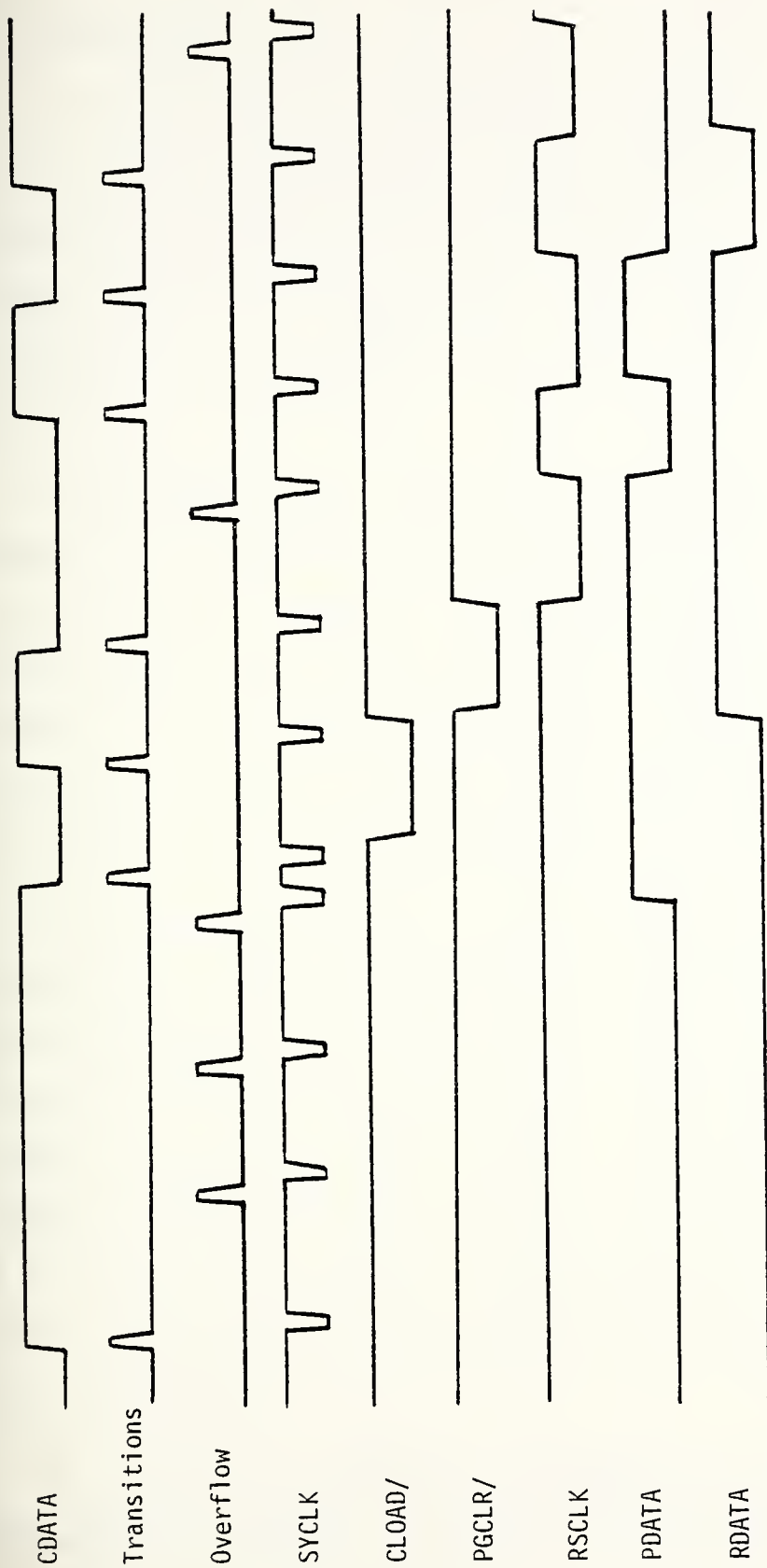


Figure 22: Sync Detector Receive Timing - Command Sync, Leading 1

C. CONTROLLER

The controller is a programmable sequencer which replaces a great deal of random logic to generate the control signals needed by the receiver and transmitter. The controller, shown in figure 23, consists of a pair of 74LS161 counters (A16 and A17), a 2708 EPROM (B18), and a pair of 74LS175 flip flops (B16 and B17).

The desired control signals are programmed into the memory and the counters are loaded to the starting address of the program. Programs are written for transmit and receive sequences for both command and data messages. The counters increment the ROM address once each 2 MHz clock and count until the stop bit out of the ROM goes high. The counters stay at this address until they are reset.

The ROM provides signals for the receiver shift register clock (RSCLK), the transmitter shift register clock (TSCLK), and the encoder. The encoder control signals are the encoder clock (ENCOD), the parity gate (PGATE), the word gate (WGATE), and the sync gate (SGATE/). The ROM outputs are clocked through flip flops B16 and B17 to eliminate glitches on the control lines and to provide true and inverted signals. During receive, RSYNC indicates the sync type of the received message.

In the initial hardware, the EPROM was programmed for 16 bit messages. Using a 1K memory, four complete sets of trans-

mit and receive programs for both sync patterns can be stored. This allows 64 addresses for each of the 16 individual routines. The receive sequence for a command message begins at ROM address 000 (hex) and begins at 03F for a data message. Transmit sequences begin at 080 for command messages and 0C0 for data messages. Additional program sets would begin at 100, 200, and 300 (hex). If 32 bit words are desired, 128 ROM addresses per program are required, limiting the system to a maximum of two sets of programs. The controller program used in the initial design is shown in table 1.

The circuitry can be easily modified to accommodate a 2716 type 2K EPROM which will allow up to four sets of 32 bit programs to be stored. To modify the controller, remove the +12 and -5 volt supply from the 2708. Connect FMAT1 to the A10 input and connect FMAT0 to the A9 input of the 2716. Move TMODE from pin 6 of A17 to pin 23 of the 2716. Ground pin 5 of A17 and connect SYMOD to pin 6 of A17. Finally, move RSYNC to pin 11 of A17.

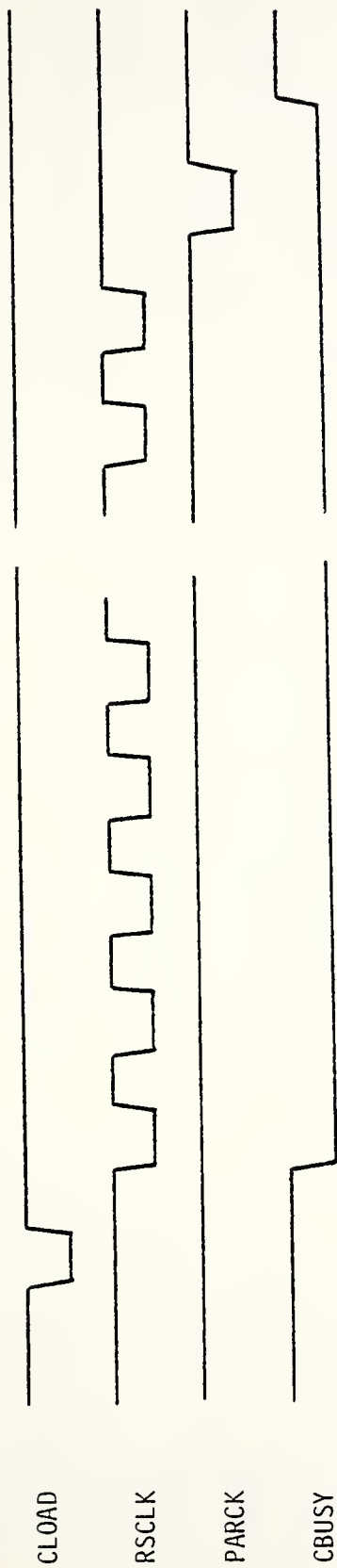


Figure 24: Controller Receive Sequence

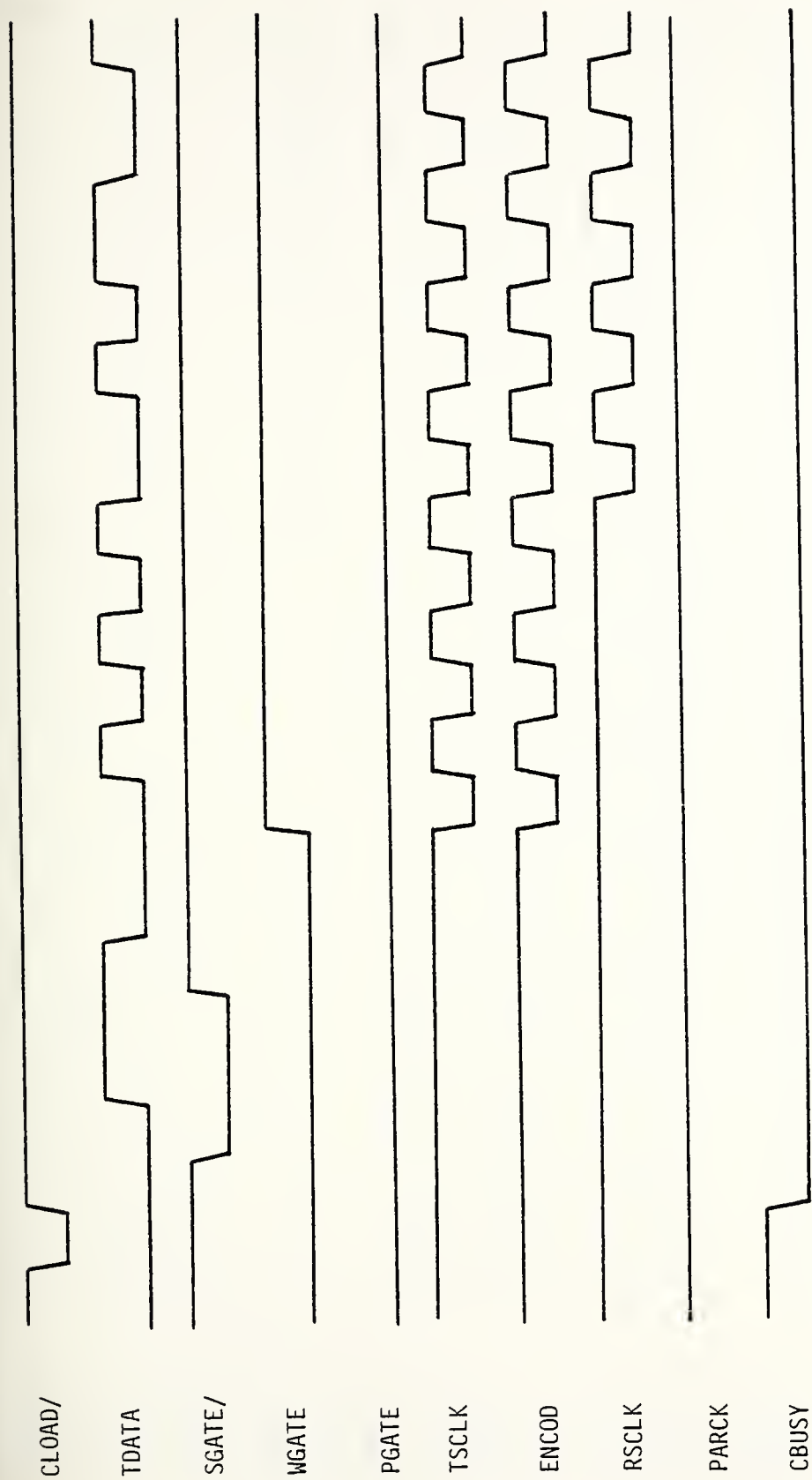


Figure 25: Controller Transmit Sequence - Data Sync (Part 1)

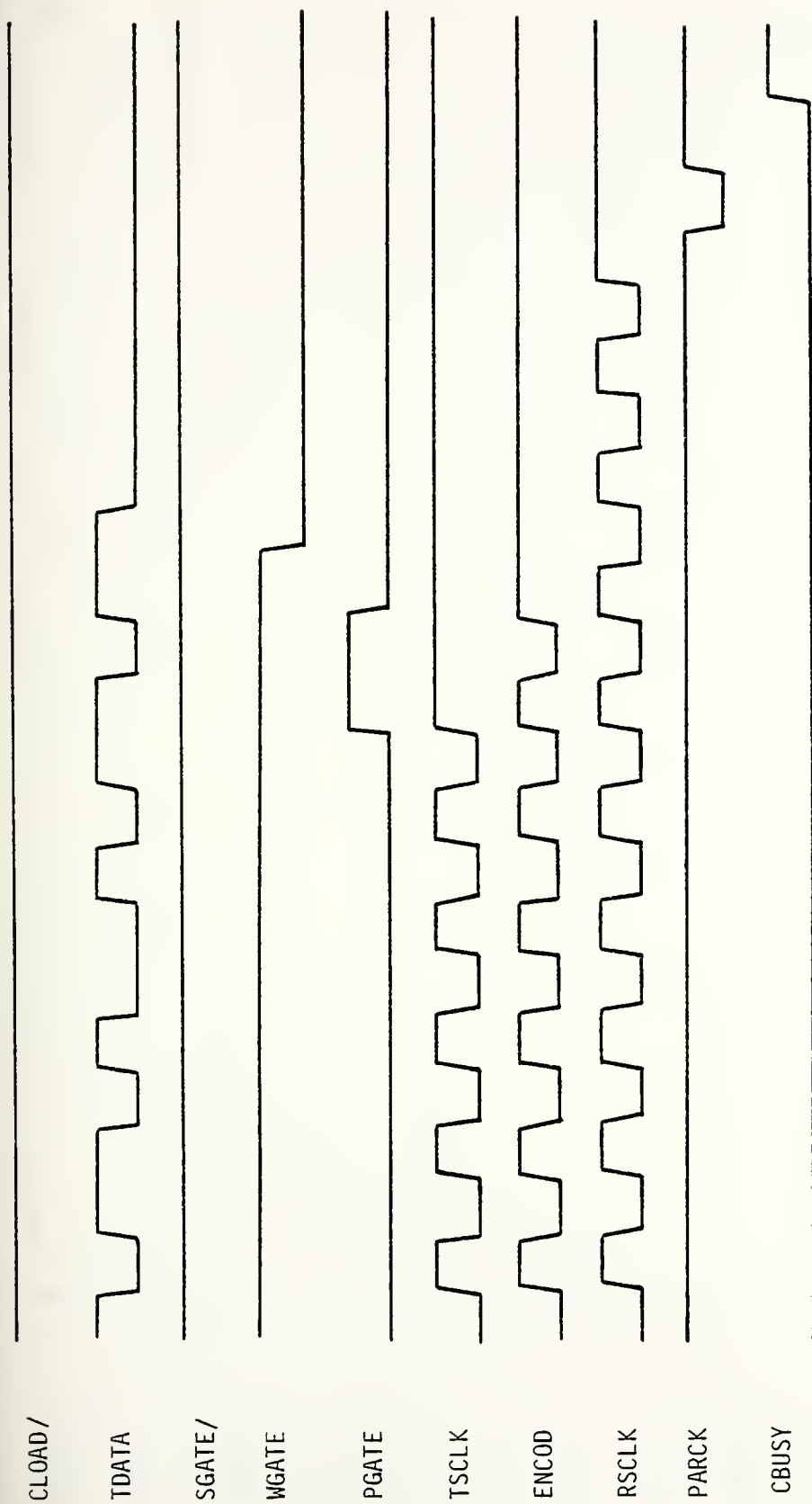


Figure 26: Controller Transmit Sequence - Data Sync (Part 2)

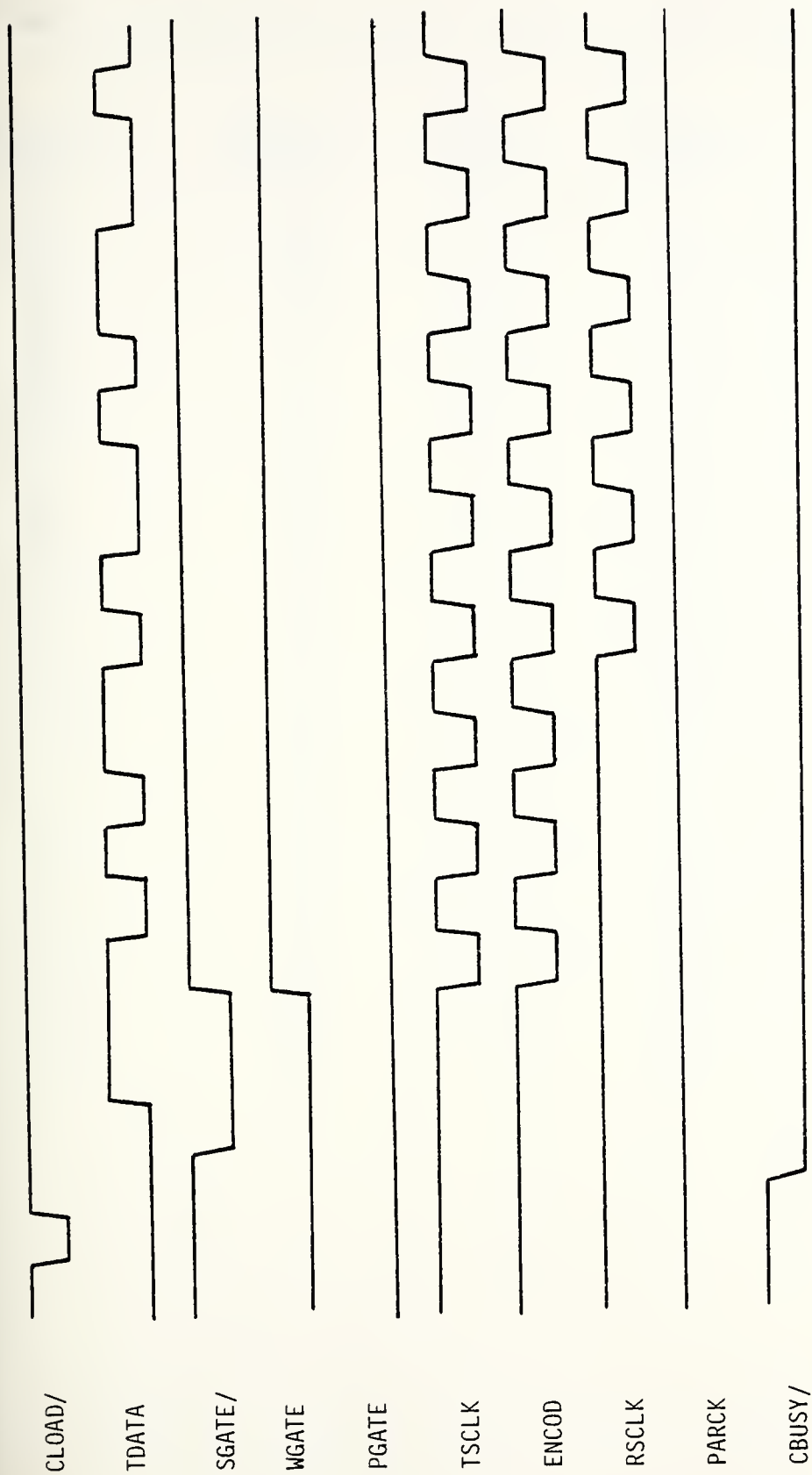


Figure 27: Controller Transmit Sequence - Command Sync

Table 1: Serial Interface Controller ROM Program

ROM Address (hex)	Receive Command	Receive Data	Transmit Command	Transmit Data
00	7E	7E	3F	3F
01	7F	7F	3F	3F
02	7E	7E	3F	3F
03	7F	7F	53	7F
04	7E	7E	5F	7F
05	7F	7F	53	7F
06	7E	7E	5F	53
07	7F	7F	53	5F
08	7E	7E	5F	53
09	7F	7F	52	5F
0A	7E	7E	5F	53
0B	7F	7F	52	5F
0C	7E	7E	5F	52
0D	7F	7F	52	5F
0E	7E	7E	5F	52
0F	7F	7F	52	5F
10	7E	7E	5F	52
11	7F	7F	52	5F
12	7E	7E	5F	52
13	7F	7F	52	5F
14	7E	7E	5F	52
15	7F	7F	52	5F
16	7E	7E	5F	52
17	7F	7F	52	5F
18	7E	7E	5F	52
19	7F	7F	52	5F
1A	7E	7E	5F	52
1B	7F	7F	52	5F
1C	7E	7E	5F	52
1D	7F	7F	52	5F
1E	7E	7E	5F	52
1F	7F	7F	52	5F

TABLE 1: Serial Interface controller ROM Program (Cont'd)

ROM Address (hex)	Receive Command	Receive Data	Transmit Command	Transmit Data
20	7E	7E	5F	52
21	7F	7F	52	5F
22	7D	7D	4F	52
23	7F	7F	46	5F
24	7F	7F	5F	52
25	7F	7F	7E	4F
26	7F	7F	7F	46
27	FF	FF	7E	5F
28	FF	FF	7F	7E
29	FF	FF	7E	7F
2A	FF	FF	7F	7E
2B	FF	FF	7D	7F
2C	FF	FF	7F	7E
2D	FF	FF	7F	7F
2E	FF	FF	7F	7D
2F	FF	FF	7F	7F
30	FF	FF	FF	7F
31	FF	FF	FF	7F
32	FF	FF	FF	7F
33	FF	FF	FF	FF

D. TRANSMITTER

The transmitter consists of the parallel-to-serial shift register, the transmitter parity generator, and the encoder. The shift register is composed of 74LS165 eight bit shift registers, C13 through C16. Data is recirculated in the shift register to provide an automatic reload of the previous message. The shift registers are loaded by the CPU by applying a low level to the load controls, STBL0/ through STBL3/. The transmitter receives 16 clocks (TSCLK) from the controller. In the basic 16 bit configuration, only C15 and C16 are used. To expand to 32 bits/word, additional 74LS165's need to be installed in C14 and C13. The sockets for these additional shift registers are provided and are prewired.

The encoder generates the Manchester code from the NRZ data out of the parallel-to-serial shift register. It has provisions for generating the non-Manchester sync characters in addition to inserting the parity bit at the end of the data. NAND gates C18, C19 and flip flop D19 form the encoder. The gates form a multiplexer to select the flip flop input from the data, parity bit and control signals.

To generate the sync character, both the word gate and the parity gate control lines are held low, forcing the output of the associated gates high. The flip flop output is forced high by applying the active low sync gate to the second half of gate C19. The sync gate is applied for three

2 MHz clock periods, corresponding to one and a half bit times.

The data is sent by removing the sync gate and enabling the word gate while holding the parity gate low. This enables the top gate of C18 and the first half of C19. The encoder clock, a 1 MHz square wave is now applied to the encoder. When the encoder clock is high, data present on the output of the shift register is passed to the flip flop input, generating the first half of the Manchester bit. When the clock goes low, the inverse of the last data output is applied to the flip flop and the bit is completed. This process repeats 16 times.

At the end of the data word, the word gate is removed and the parity gate is applied to the encoder. This switches the flip flop input from the shift register to the parity generator output. The parity bit is added to the message by the same process used for the previous 16 bits. At the end of this last bit interval, all control signals are removed, the flip flop output returns low, and the message is completed.

The parity generator, C17, is a 74LS161 counter connected to the output of the parallel-to-serial shift register. The counter is cleared before each transmission by PGCLR/ from the sync detector. The asynchronous clear input is used in lieu of the load control because no clock is available at the time the counter is cleared. The parity generator receives the same clock as the shift register.

At the end of the sixteenth clock, the LSB of the counter specifies the parity bit to be sent during the next bit time. The counter output can be inverted by a 74LS86 exclusive-or gate to provide even or odd parity. Parity is taken over the entire 17 bit message. If odd parity is selected, the total number of ones in the 17 bits must be odd. If the LSB of the counter is zero, signifying that the 16 data bits had an even number of ones, the LSB is inverted and added as a parity bit so that the total is odd. If the counter output had been a one, the 16 bits already had an odd number of ones and no additional one is required in the parity bit. The reverse is true for even parity; the LSB of the parity generator is applied without inversion.

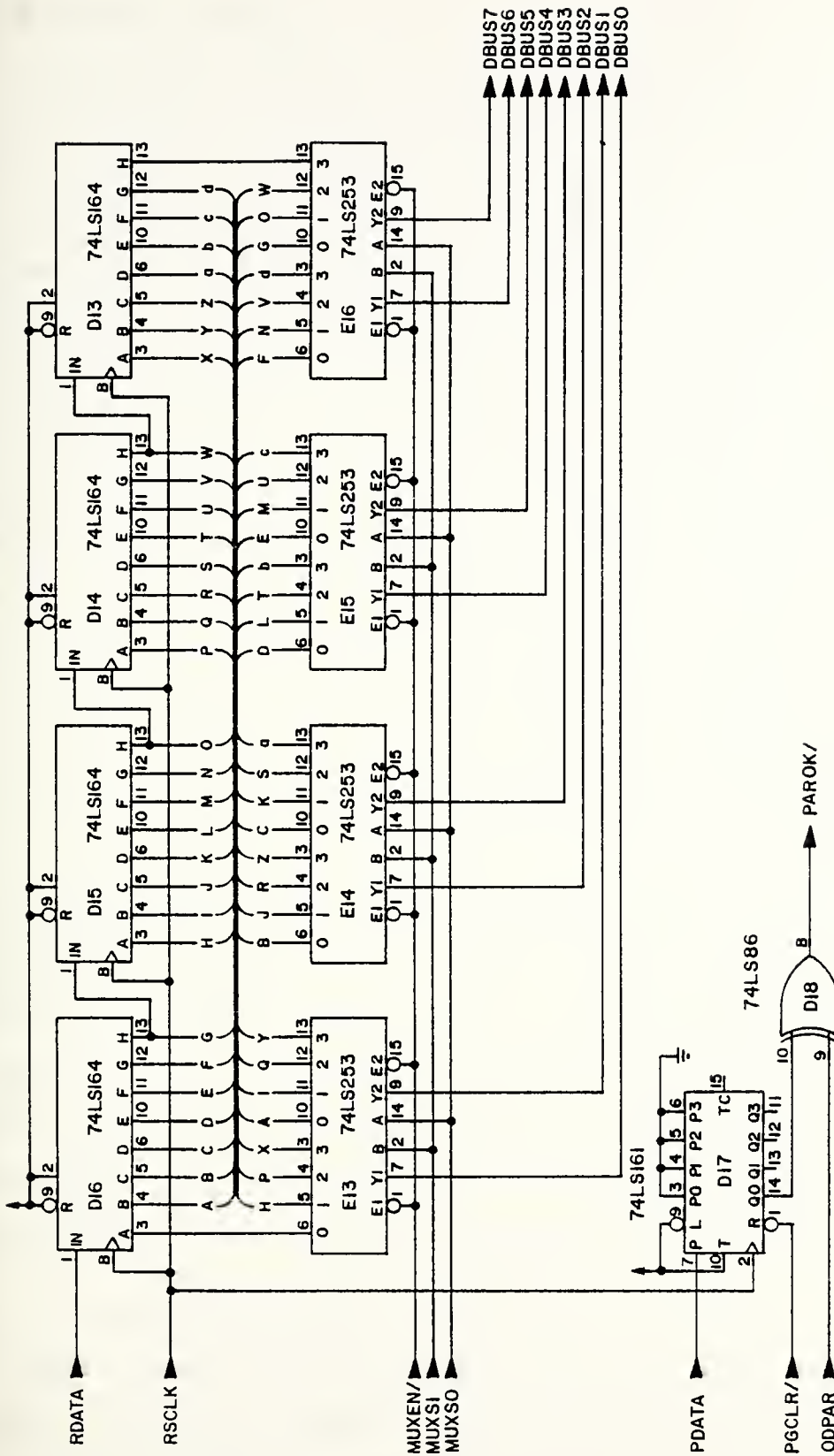
E. RECEIVER

The receiver is the serial-to-parallel counterpart of the transmitter and consists of a shift register and a parity counter. The shift register is composed of eight bit 74LS164's, and sockets are provided to accommodate 32 bit word lengths.

Due to the limited number of control signals available from the controller, the shift register and parity counter are activated by the same shift clock (RSCLK). Since the counter must receive 17 clocks to compute the parity of the entire message, the shift register receives one extra clock. This is compensated by delaying the shift register data (RDATA) by two 2 MHz clocks, or one bit time, with respect to the parity counter input (PDATA). The first bit into the receiver shift register is actually the tail end of the sync character, but after 17 clocks, the shift register contains the proper data.

The parity counter, D17, is cleared at the start of each received message by PGCLR/. At the end of the receive sequence, the parity check is made available to the CPU and the receiver interrupt flip flop is set by PARCK. The output of the counter is passed through an exclusive-or gate, D18, to allow selection of even or odd parity by the CPU control ODPAR. If no detectable errors occurred, the signal to the CPU is low.

The outputs of the shift registers are connected to the CPU data bus via four 74LS253 tri-state dual four input multiplexers, E13 through E16. The tri-state feature allows the outputs to be connected directly to the bus. The received bytes are read by selecting the desired data byte through the multiplexer select inputs (MUXS0 and MUXS1) and applying an active low output enable (MUXEN/). The use of tri-state shift registers would have reduced the parts count of the receiver but were not readily available at the time of fabrication.



RECEIVER

Figure 29: Receiver Schematic

F. MULTIBUS INTERFACE

The Multibus interface provides the latches, drivers, and bus controller needed to communicate with peripherals via the parallel bus. Address bits and output data are latched in three 8212 tri-state latches, E8 through E10. These latches provide the necessary drive capability for the bus and reduce the package count of this circuit. The latches are non-inverting, hence the data and address must be complemented by the CPU before any transaction. Memory read operations use a pair of 8098 tri-state drivers (E6 and E7) connected across the data output latch. Since these devices are inverting, no inversion is required in the CPU. Room is available on the serial interface board for additional latches and drivers to adapt the board to a 16 bit data bus, although no circuitry is provided.

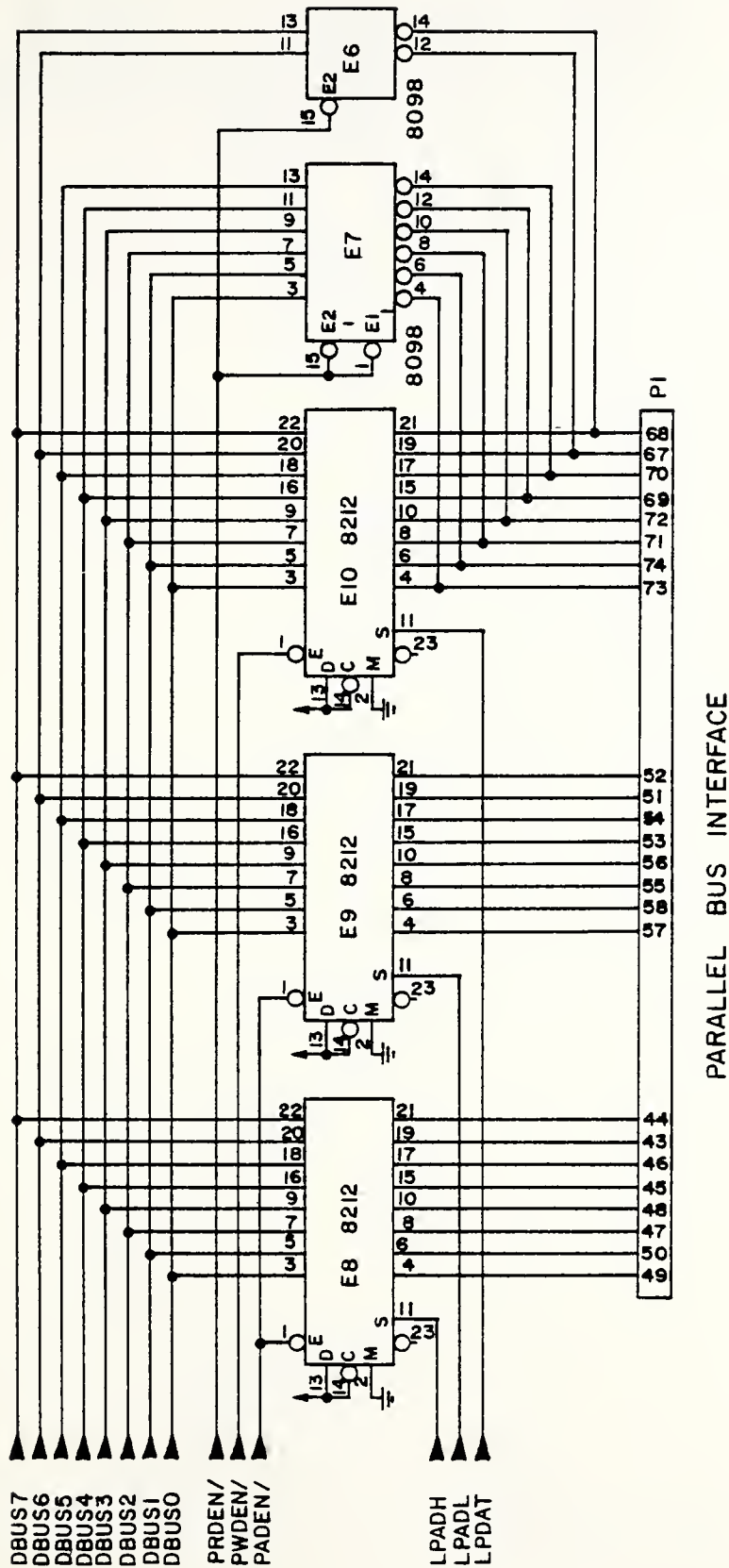
The Multibus controller is a minimal bus controller requiring CPU interaction to function. The parallel bus controller can only communicate with external memory and no single master facilities, such as bus clock or initialization signal, are provided. Since at least one board on the parallel bus will have these facilities, their duplication in the serial interface hardware was not necessary.

The parallel exchange begins with a CPU request, PTREQ/. This active low signal is applied to the bus request (BREQ/) and the bus priority output (BPRO/) lines after the next bus clock (BCLK/). The bus request line is used in a parallel

priority resolution scheme, such as a priority encoder, while the bus priority out is used in a serial bus priority system. The bus arbitration logic is located on a monitor board in the MDS system.

When a board's bus priority input (BPRN/) goes active and after the current user has released the bus busy (BUSY/) line, the controller sets flip flop C2 to place it in the bus control mode. One bus clock after the flip flop was set, the BUSY/ line is pulled low and the address and data drivers are turned on by PADEN/ and PWDEN/. After another bus clock, the read or write command (MRDC/ or MWTC/) is issued in accordance with the sense of the CPU read/write select lines RDWTS and the transfer begins. The CPU waits until the transfer acknowledge (XACK/) is received from the external device before ending the transaction. In the case of a memory read, the data input drivers are enabled by PRDEN/ and the CPU reads the data.

The transaction ends when the CPU removes the transfer request signal. The read or write command is removed first, followed one bus clock later by the removal of the address, data, and the BUSY/ signal. The bus is then released to the next user.



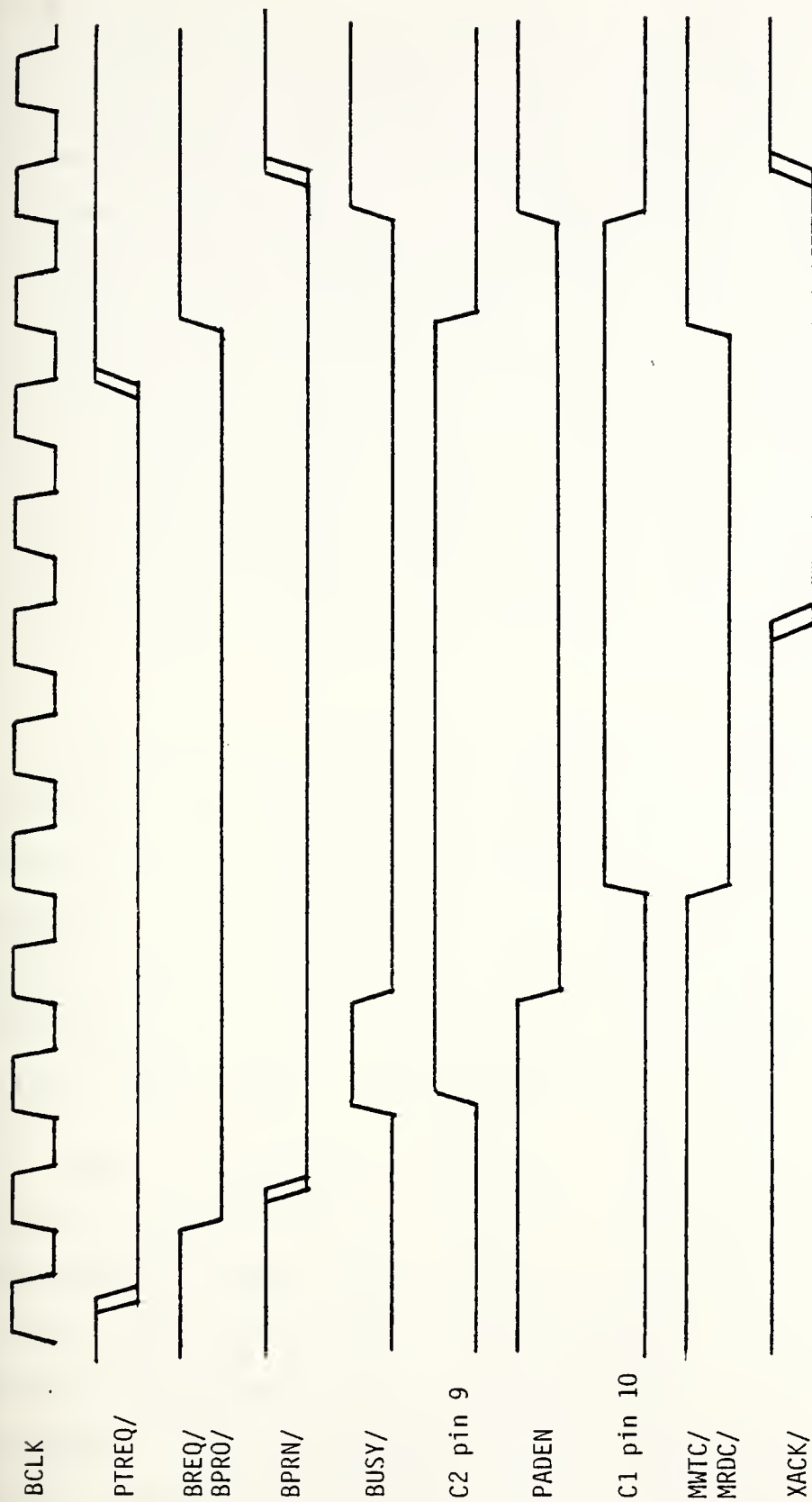


Figure 32: Multibus Controller Timing

G. CPU GROUP

The serial interface board utilizes the Intel 8748 single chip computer as its central processor. This chip contains 64 bytes of internal read/write memory and 1K bytes of electrically programmable/UV erasable read only memory. The 8748 supports 27 input/output lines consisting of two eight bit ports, an eight bit data bus port, two test inputs and an external interrupt input. The chip includes an internal timer/counter and includes facilities for external instruction fetches and single stepping. For additional information on the 8748 the reader is referred to the MCS-48 Users Manual.

The CPU group is composed of the 8748 single chip computer and its associated latches, decoders, and gates. The 8748 derives its clock from the constant clock (CCLK/) provided by the Intel Multibus. This 9.2 MHz clock is divided by flip flop C2 to run the CPU at a nominal 4.6 MHz. The 8748 RESET line is connected to the Multibus initialization signal (INIT/) to provide power up reset.

The T0 test input is connected to the Multibus XACK/ line and is used to tell the CPU when the parallel bus transaction is completed. The T1 input comes from the controller CBUSY/ line in the serial hardware and can be used for a similar purpose but is not utilized in the initial software. The interrupt input comes from the flip flop D19 and informs the CPU that a message has arrived.

All lines of port 1 are dedicated to control functions on the serial interface board. These lines are all outputs. Two inputs to port 2 inform the CPU of the sync type of the received message (RSYNC) and the results of the parity check (PAROK/). Four other lines are wired to a switch register for manual inputs but are not used in the initial software. During operations with external memory and during the single step mode, the lower four lines of port 2 contain the upper four bits of the internal twelve bit program counter. The bus port is operated as a true dibirectional data port with external bus drivers (D5 through D7) to increase the drive capability. All data into and out of the 8748 passses through this port.

External latches are addressed by the MOVX instruction in the 8748. During this operation, an eight bit address is issued in advance of the data on the bus port. An address latch enable (ALE) pulse latches the address into an external register (B5 and B6). The lower four address bits are decoded by C6 and used as load controls for the transmitter shift registers and parallel bus latches. The only external data inputs to the CPU come from the receiver multiplexers and the parallel bus data input buffer. These are selected by bit 3 of the address register and enabled by the CPU read strobe. The receiver multiplexers receive address bits (MUXS0 and MUXS1) from the lower two bits of the address register.

An external 1K by 8 bit program memory (B3 and B4) is provided for debugging purposes. The ten bit address for the memory is supplied by eight bits from the address latch and two bits from port 2. The memory data output is applied directly to the CPU data bus port, bypassing the bus drivers. The memory chip select is provided by the 8748 PSEN signal.

The external memory is loaded through a sixteen pin connector, B2, and an interface cable to an SBC 80/20 computer. The eight data lines are multiplexed with the ten address lines. An external address latch enable, write enable, and chip select are also supplied through the cable.

A switch register on the serial interface board selects the load/run mode for the memory. In the load position, the chip select input is removed from the 8748 PSEN line and applied to the external chip select. The address register clock input is switched from the CPU to the external ALE. The processor is forced into reset and the external access mode is disabled allowing the data bus to float. The address and data are applied to the bus port and lower two bits of port 2, loading the program into memory. In the run position, the chip select and address latch control are returned to the CPU, the external address and data drivers are turned off, and the 8748 resumes control of the data bus.

The 8748 supports a single step mode of operation and the necessary external circuitry is provided on the serial

interface board. Flip flop B7 single steps the processor by allowing one ALE pulse before stopping the processor. In the halted state, the contents of the program counter are made available on the bus port and the lower four bits of port 2. These lines are read by the SBC 80/20 through the interface cable and displayed on a CRT terminal. With the debugging program written for the 80/20, depressing any key on the terminal will advance the 8748 to the next address. Pressing the BREAK key will step the 8748 through its program at a rapid rate.

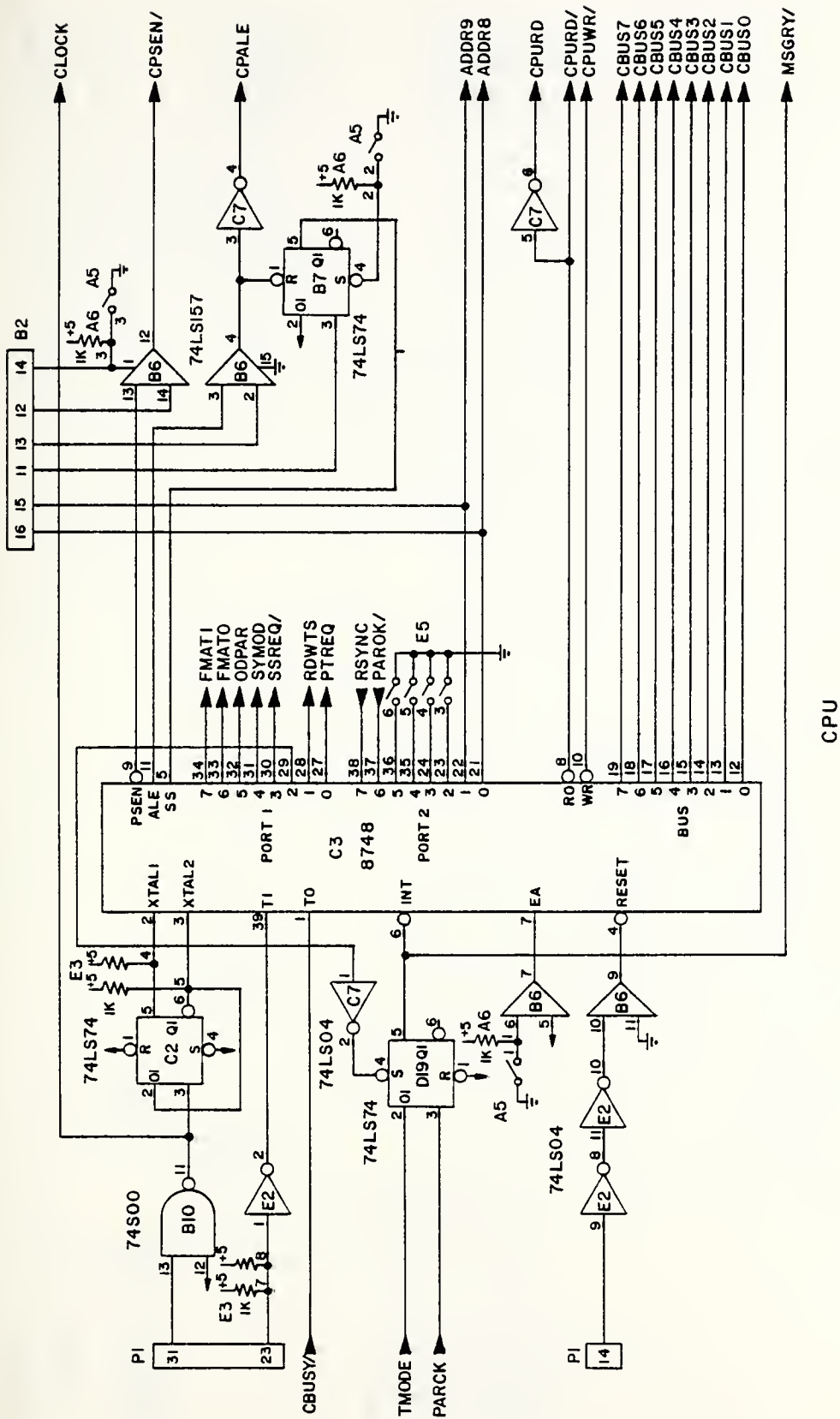


Figure 33: CPU Schematic (Part 1)

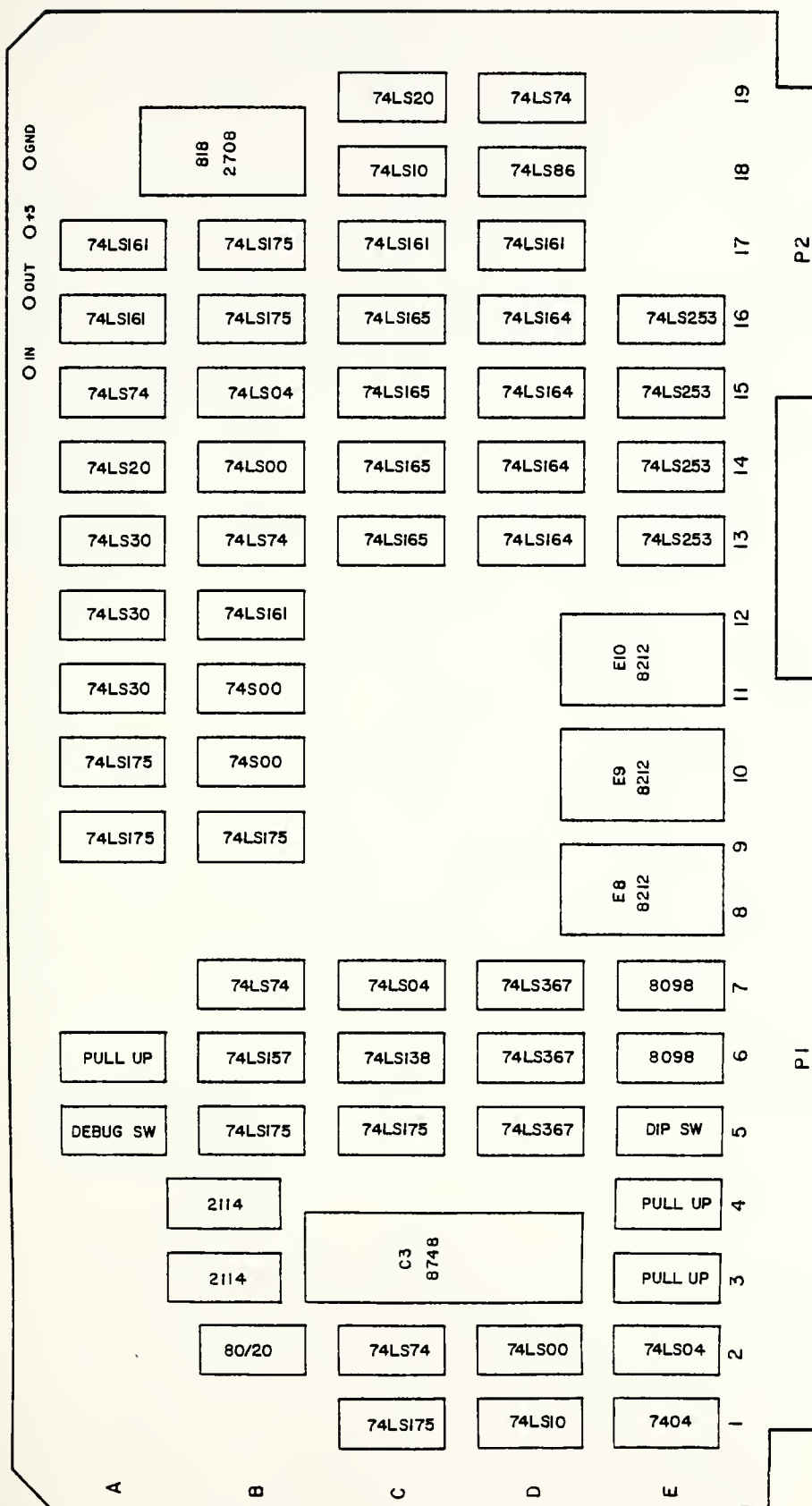


Figure 35: Component Placement

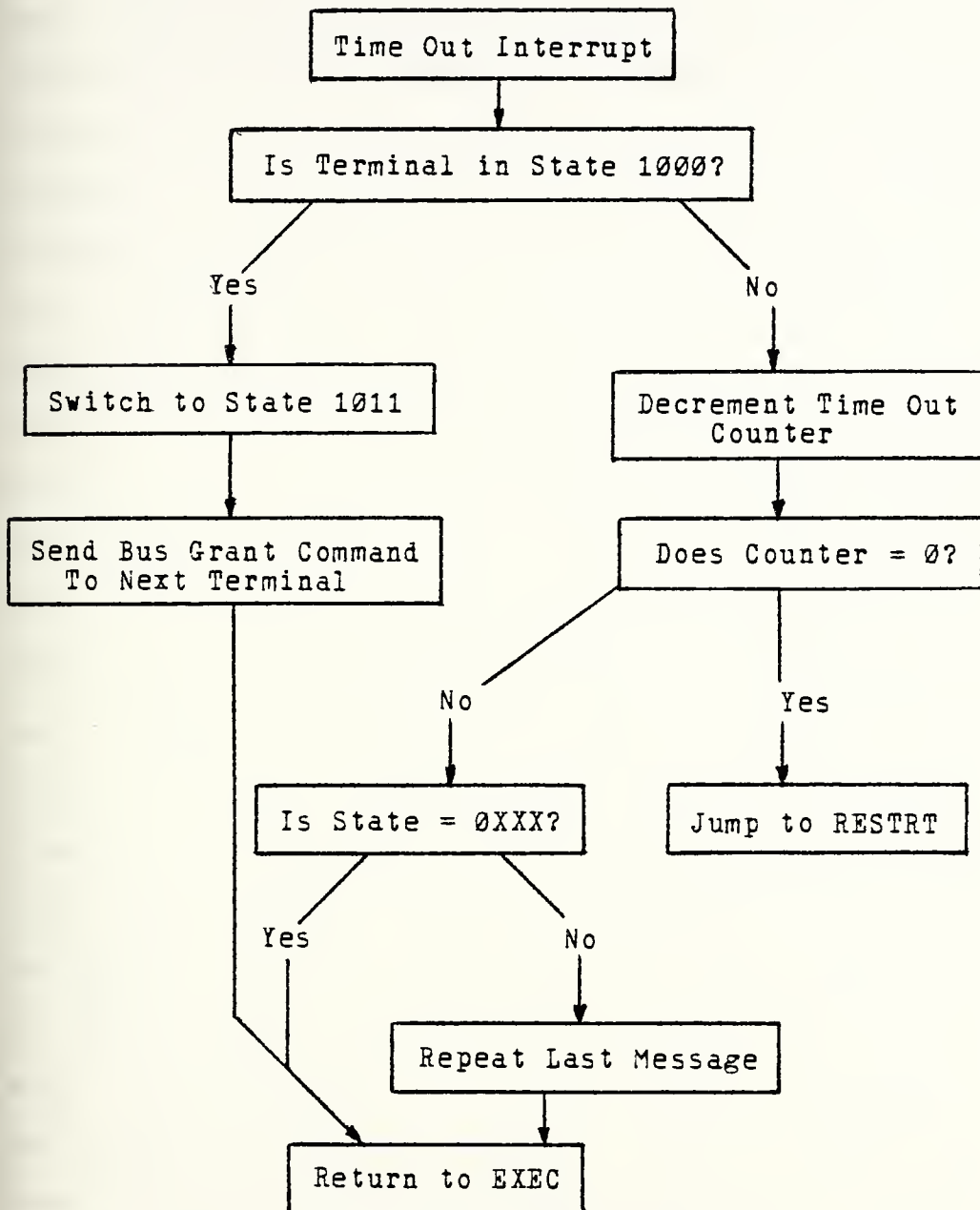
APPENDIX B

SOFTWARE REFERENCE MANUAL

A. TIME OUT INTERRUPT HANDLER - TIMEOUT

This routine services the time out interrupts generated by the time/counter. The accumulator is saved on the stack and the register bank is switched to preserve the state of the machine. If the terminal is in state 1000, waiting for a clear-to-send acknowledge, the terminal assumes the message can not be sent and switches to state 1011 to relinquish the bus. If the terminal is in one of the other bus control states, state 1XXX, the error counter is decremented and the last message is repeated. If the terminal is in a listener state, 0XXX, the only action taken is to decrement the error counter. The error counter is reloaded each time a message is received from the bus. When the error counter reaches zero, the routine vectors to address 000 and the restart routine is entered. In the present configuration, the time out interval is 10msec. and 16 consecutive timeouts cause a restart.

Figure 36: Time Out Routine Flow Chart



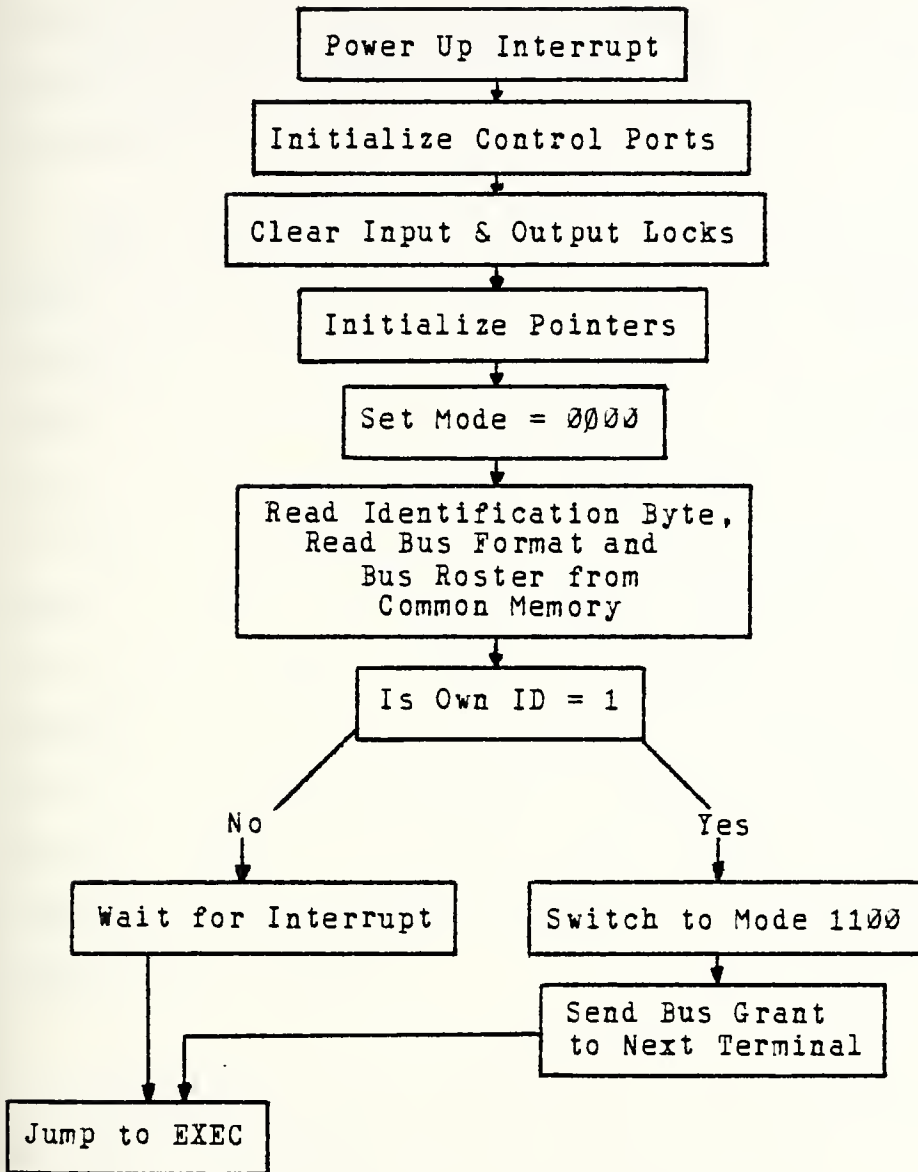
B. RESTART ROUTINE - RESTRT

This routine initializes the 8748's internal registers and flags on power up and after timeout restarts. The stack pointer is zero'ed on power up but a separate instruction is included to zero the stack pointer after time out generated restarts. The terminal state is set to 0000. The restart routine reads the terminal initialization information from the corporate memory. This consists of the bus format, an identification byte, and the bus roster. The bus format specifies one of four possible message formats and selects even or odd parity. The message format is carried in bits 6 and 7, the parity selection is done by bit 5. The lower four bits of the identification byte determine the terminal's identification number and the upper four bits specify the next terminal to receive bus control. The identification byte is used as a command message header for bus grant commands. The bus roster is an eight bit quantity that specifies which computers reside on the parallel bus. Each bit represents a possible SBC number one through eight. A one in any bit position signifies that computer is on the bus. With this configuration, the total number of computers in the system is limited to eight. A second register could be used for SBC's nine through 16 but additions to the 8748 software would be required.

If the terminal has been assigned identification number one, it assumes initial bus control duties, switching to

state 1011 and issuing a bus grant command to the terminal indicated by the upper four bits of the identification byte. After completing the restart procedures, control is turned over to the executive program EXEC.

Figure 37: Restart Routine Flow Chart



C. EXECUTIVE ROUTINE - EXEC

The executive is a continual loop of fetching and delivering messages in corporate memory. Messages leaving the 8748 buffer receive priority to keep the buffer as empty as possible.

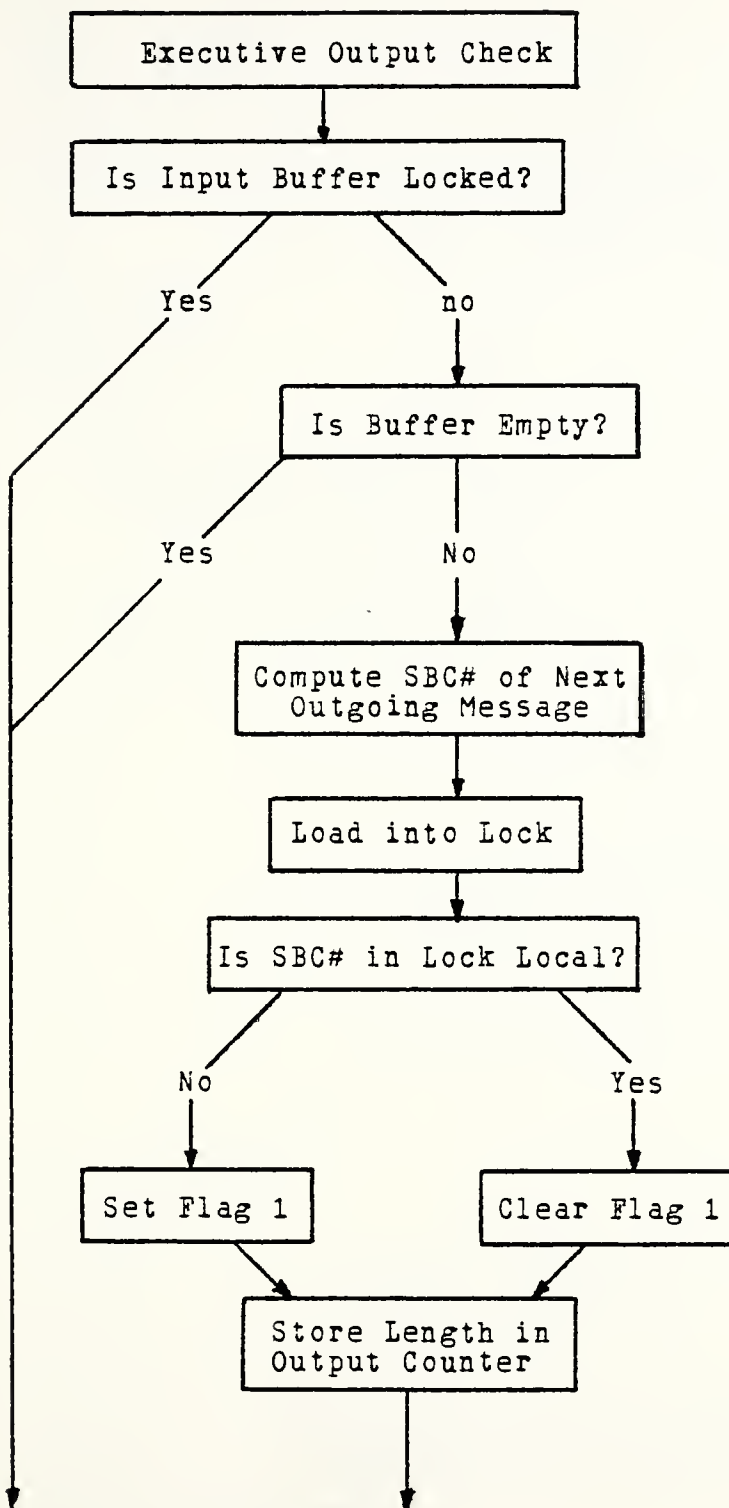
EXEC first checks the output lock of the 8748. A locked output buffer implies a serial output is in progress and no messages can be removed. If the output lock is zero, EXEC reads the receiving module number (RMN) of the next message and calculates the destination SBC number. If the message is bound for a remote computer, internal flag 1 is set to signal the serial message handling routines, and EXEC jumps to the input check. Local bound messages are delivered by SENDEXT, a routine called by EXEC. The corporate buffer is locked by the routine SETLOK and after the transfer, unlocked by UNLOCK. After delivering a parallel message, EXEC returns to the output check to try to remove another message from the 8748.

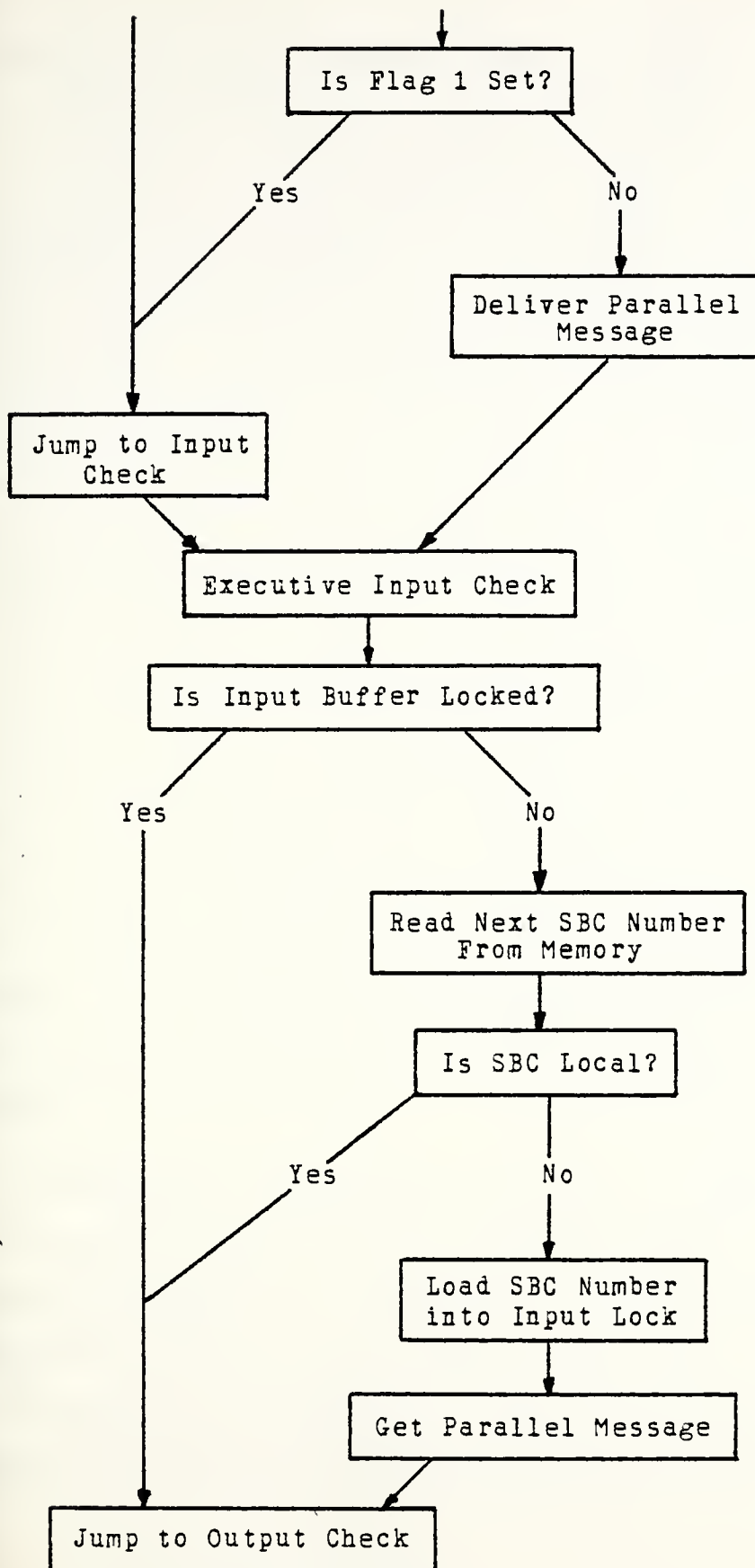
After all possible messages have been removed, EXEC looks for incoming messages in the corporate buffer. The 8748 input buffer is checked to see if a serial input is already in progress. If the buffer is zero, EXEC reads the EXTMSG register in corporate memory. If the next outgoing message in corporate memory is addressed to a computer not on the bus, EXEC calls RECEXT to move the message into the 8748 buffer.

At the end of this transaction, EXEC jumps to the output check to try to remove another message from the 8748 buffer.

External interrupts are disabled during the periods when EXEC is examining the input and output locks. This prevents a received message from locking the buffer after EXEC found it to be unlocked and began a parallel transaction. Time out interrupts do not affect any vital registers and may occur at any time during the program.

Figure 38: Executive Routine Flow Chart





D. PARALLEL MESSAGE RECEIVE ROUTINE - RECEXT

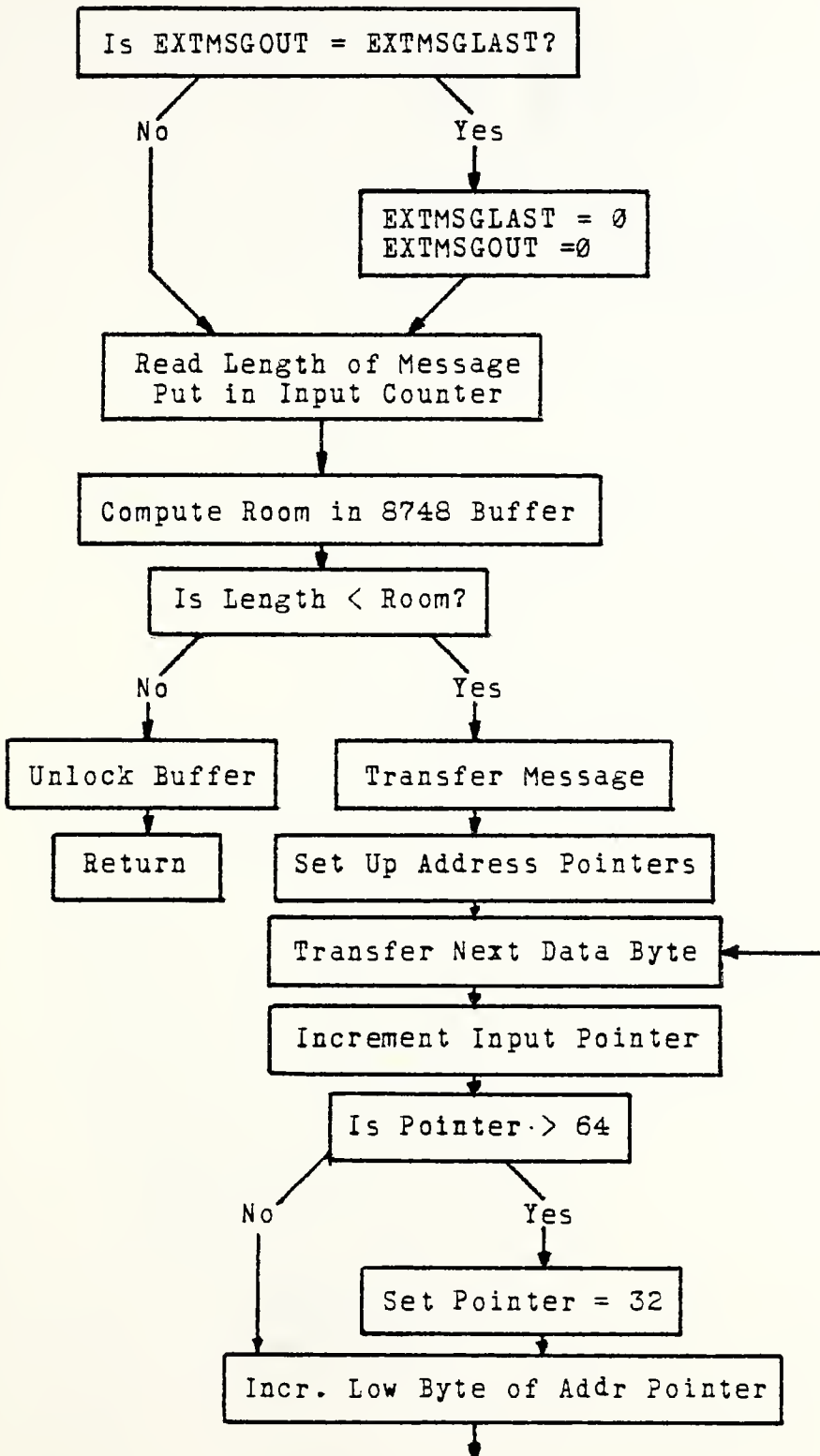
The routine is called by EXEC to transfer a message in corporate memory to the 8748 buffer. RECEXT reads the EXTMSGOUT pointer in memory to find the starting address of the next outgoing message. If EXTMSGOUT is found equal to EXTMSGLAST, the next message lies at the bottom of the buffer and EXTMSGOUT is set to zero. The length of the message is found by reading the fourth byte of the message. The room available in the 8748 buffer is calculated using the 8748 input and output pointers. If the output pointer is above the input pointer, the empty buffer lies between the two pointers and the available room is the difference in the two pointers. If the reverse is true, the empty buffer lies above the input pointer and below the output pointer. The available room is computed by subtracting the difference in the two pointers from the size of the buffer. If the length of the message is less than the available room, the message is transferred, otherwise the procedure is terminated.

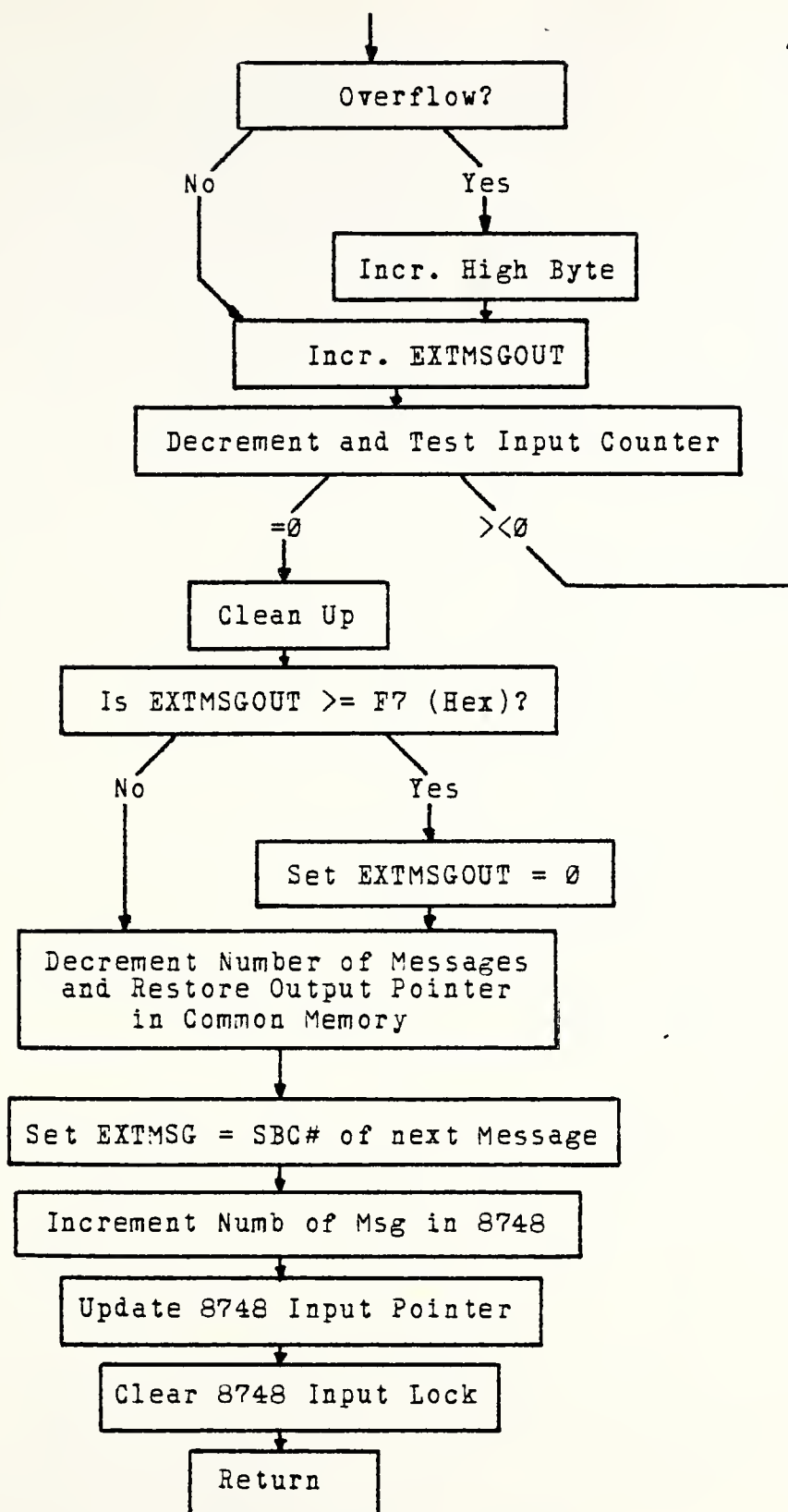
The message is transferred one byte at a time. The 8748 input pointer and the corporate output pointer are incremented with each byte transferred. If the 8748 input pointer reaches 64, the top of the buffer, the pointer is reset to 32, the bottom of the buffer. The length of the message is loaded into the input counter register in the 8748 and

decremented with each byte transferred. When the input counter reaches zero, the transfer is complete.

Before leaving this routine, the output pointer in common memory is updated and the number of messages in the buffer, NUMEXTMSG is decremented. If the number of messages remaining is not zero, RECEXT calls SETEXT to compute the SBC number of the next message and load it into EXTMSG. The 8748 message count is incremented, the input lock cleared and control returns to EXEC.

Figure 39: Parallel Message Receive Routine Flow Chart



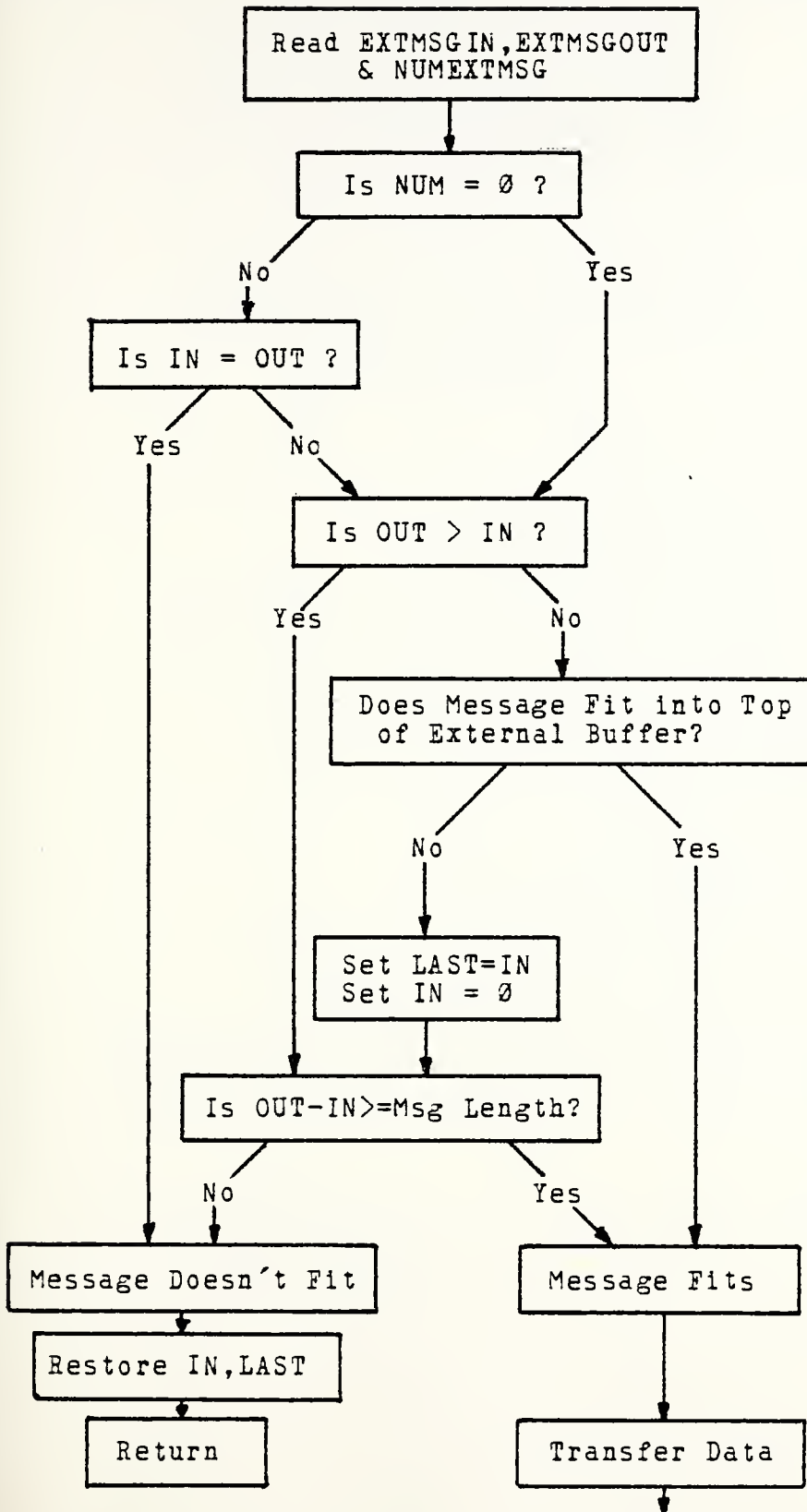


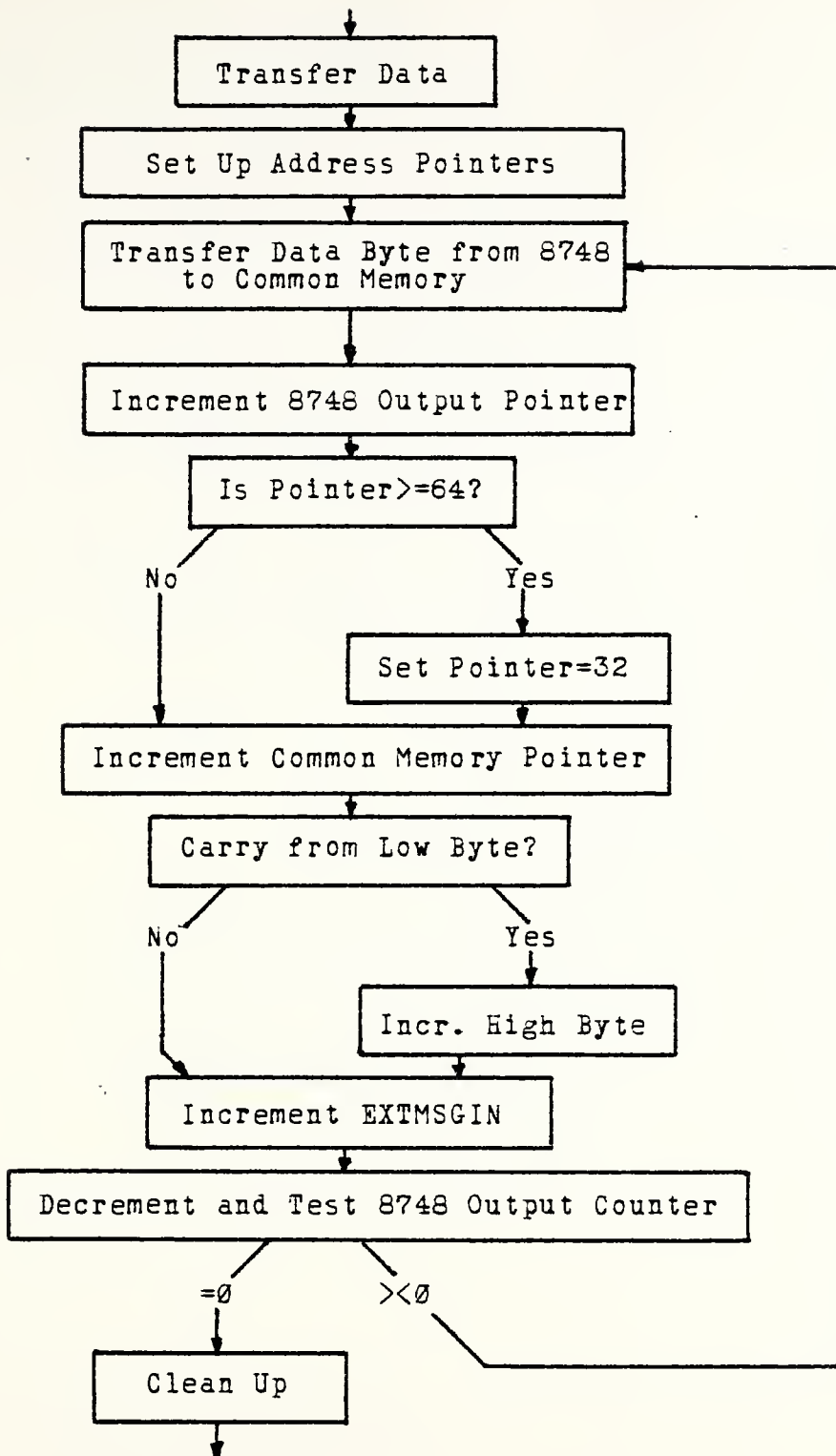
E. PARALLEL MESSAGE SEND ROUTINE - SENDEXT

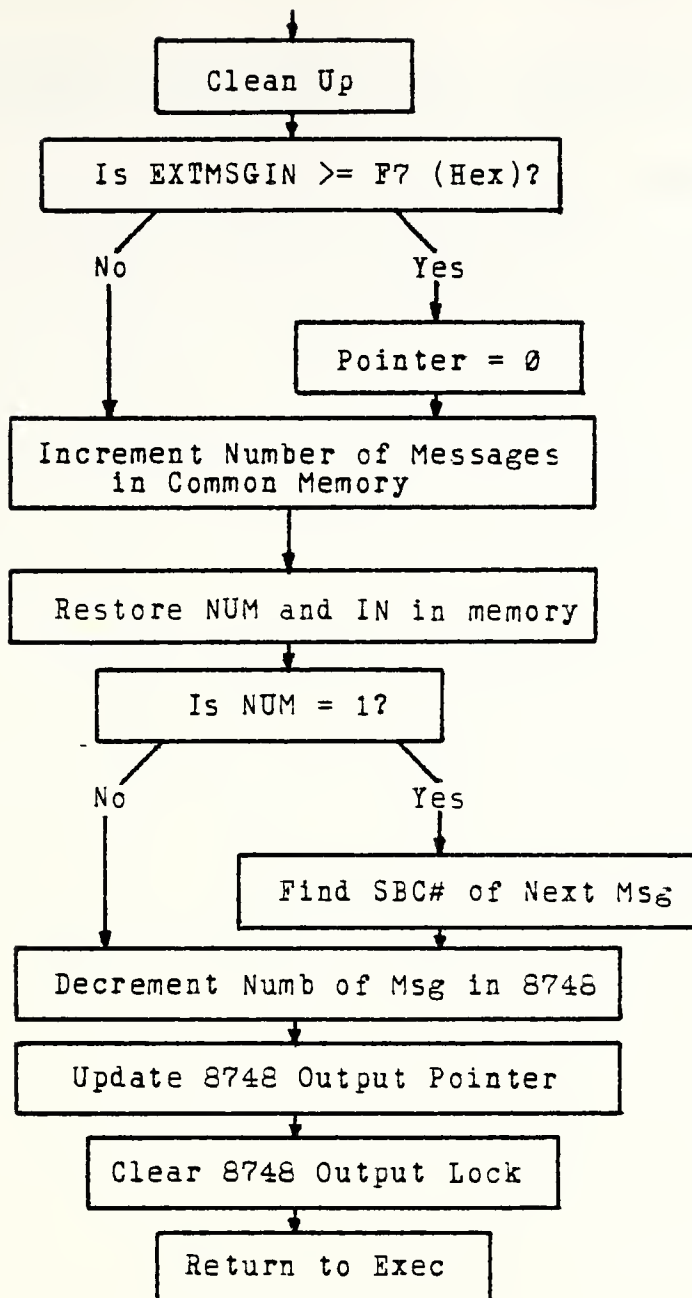
This routine is called by EXEC to deliver a message to the corporate buffer. SENDEXT first checks the available room in the corporate buffer. The input pointer, EXTMSGIN, the output pointer, EXTMSGOUT, and the number of messages in the buffer, NUMEXTMSG, are used in the room calculation. If the number of messages is not zero and the input and output pointers are equal, the buffer has overflowed and no message can be delivered. If the input pointer is above the output pointer, SENDEXT tries to fit the message into the remaining room at the top of the buffer. If insufficient room exists, SENDEXT sets the input pointer to zero and tries to fit the message into the bottom of the buffer. If the input pointer is below the output pointer, the empty buffer lies between the two pointers. If the message does not fit into the buffer, the process is aborted and control returned to EXEC. If sufficient room exists, the data is transferred.

The message transfer follows the technique used in RECEXT. The length of the message is read from the fourth byte and loaded into the output counter. The 8748 output pointer and the corporate buffer input pointer are incremented and the output counter decremented after each byte is moved. Following the transfer, the memory pointers are updated and the message counts adjusted. The 8748 output lock is cleared and control returns to EXEC.

Figure 40: Parallel Message Send Routine Flow Chart







F. PARALLEL UTILITY ROUTINES: UNLOCK, SETLOCK, AND SETEXT

These routines are called by SENDEXT and RECEXT.

UNLOCK merely clears the EXTMSGLOCK register in corporate memory. SETLOCK locks the buffer by reading and rereading the lock until the current user has released it. The desired SBC number is then written into the lock. EXTMSGLOCK is then read to make certain that another computer did not already achieve lock.

SETEXT determines the next SBC number to be loaded into EXTMSG. If the buffer is empty, NUMEXTMSG = 0, EXTMSG is cleared. If the buffer still contains messages, the receiver module number is read from the fourth message byte, divided by eight and incremented to find the SBC number. This value is then written into EXTMSG.

Figure 41: Set EXTMSG Routine Flow Chart

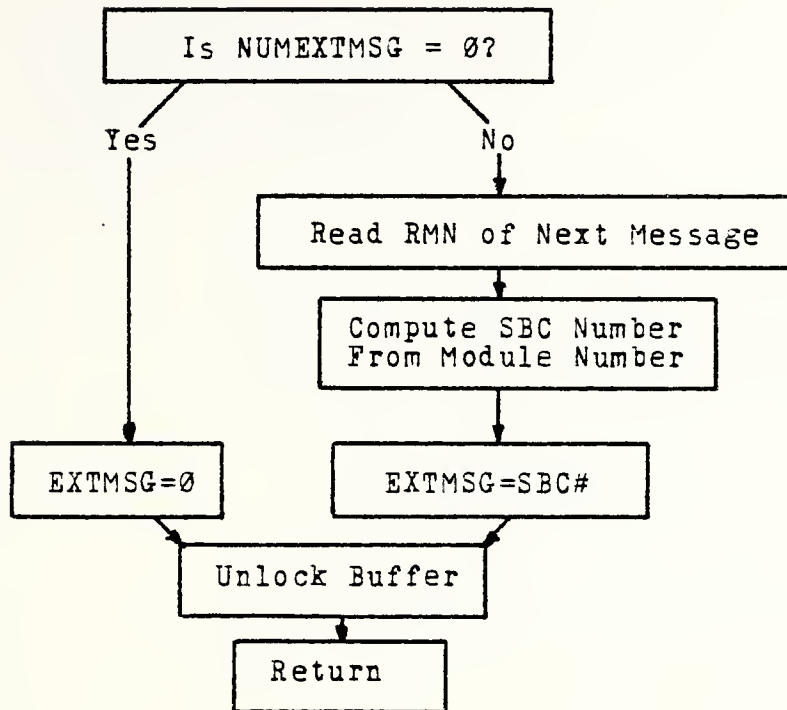
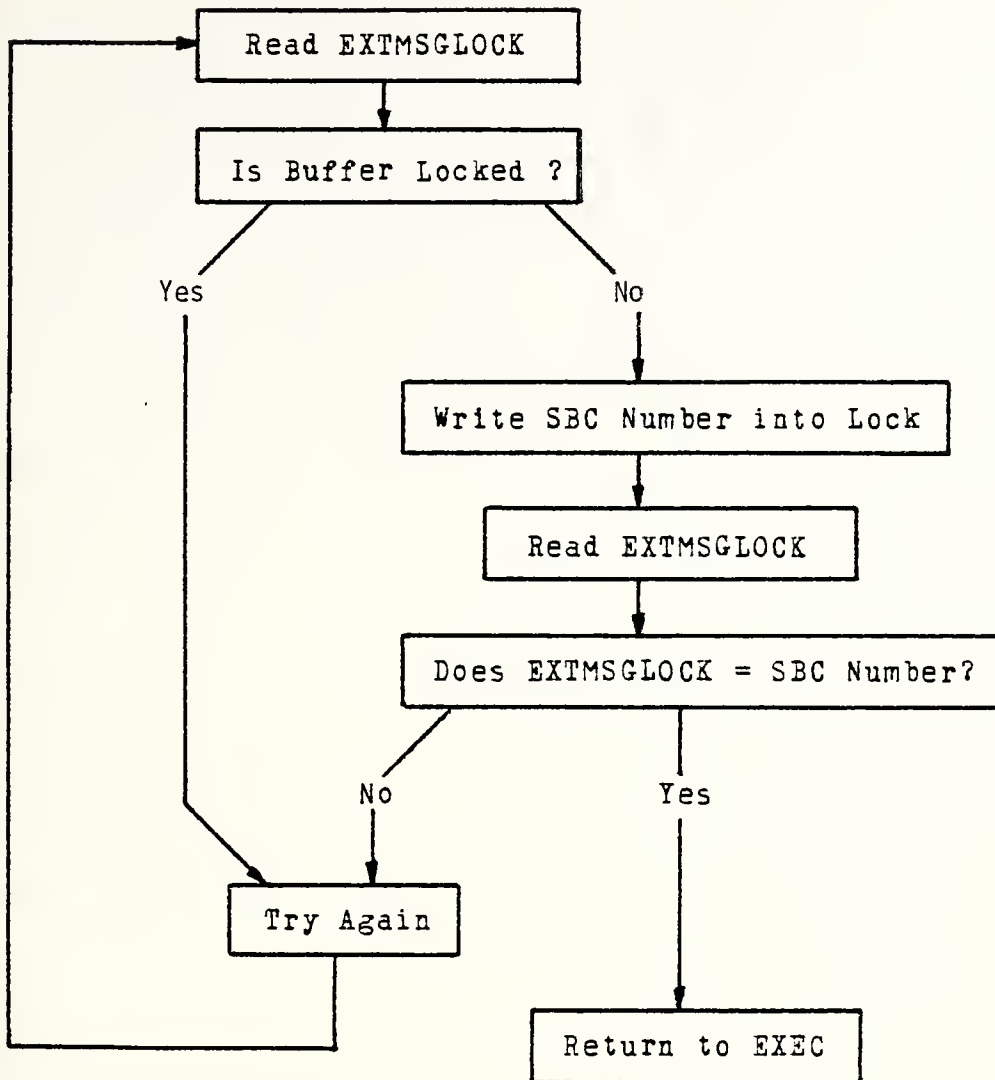


Figure 42: Message Buffer Lock Routine Flow Chart



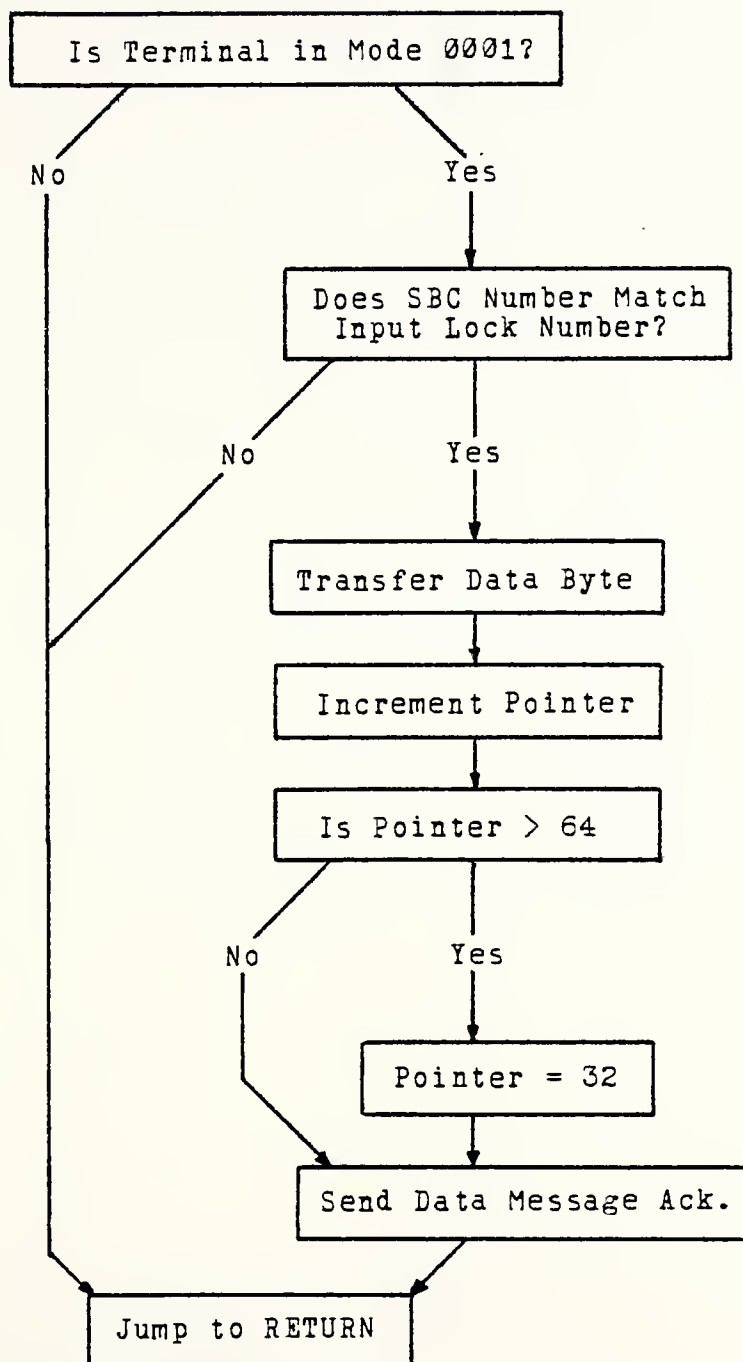
G. SERIAL UTILITY ROUTINES: SENDDO, SENDCO, SENDCI, SENDCN

These routines are called by the serial message handler to send a message on the serial bus. SENDDO is used to send data messages to another terminal. The header for the message is obtained from the 8748 output lock and contains the destination SBC in the upper four bits and the source SBC in the lower four bits. SENDCO is similar to SENDDO but is used to send a command message to the SBC specified by the output lock. This routine is used to send request-to-send and end-of-message commands. SENDCI sends a command to the SBC number specified by swapping the nibbles of the 8748 input lock and is used to send acknowledgements for clear-to-send, end-of-message, and data messages. SENDCN is used to send a bus grant to the next terminal to receive bus control. SENDCN gets the message header from the identification byte stored in the 8748.

H. SERIAL DATA MESSAGE RECEIVE ROUTINE - RECMSG

RECMSG checks the terminal state for the message receive mode, 1000, and checks the message source against the SBC number in the input lock. If both agree, the message is assumed correct and the data byte is loaded into the 8748 buffer at the current position of the input pointer. The input pointer is incremented and checked for overflow. A data acknowledgement is issued and control is passed to the RETURN routine which cleans up before ending the interrupt sequence.

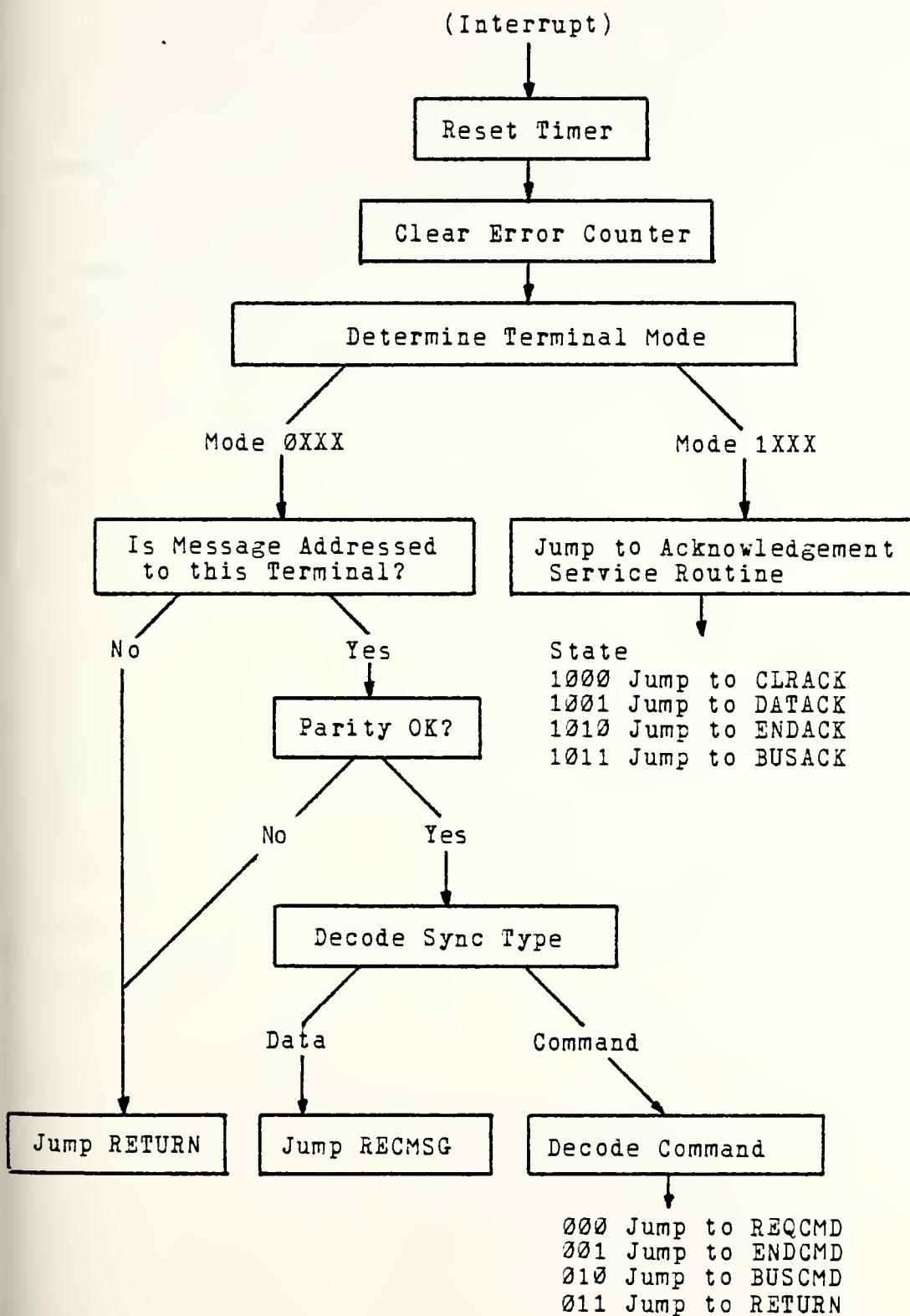
Figure 43: Serial Data Message Receive Routine Flow Chart



I. SERIAL MESSAGE DECODER ROUTINE - MSGDEC

This routine services the external interrupt caused by the reception of a serial message. In response to the interrupt, MSGDEC switches register banks and pops the accumulator on the stack. If the terminal is in a bus control mode, state 1XXX, the received message is considered an acknowledgement of the last command regardless of its content. Based on the state of the terminal, the proper command acknowledge routine is called. If the terminal is in a listener mode, state 0XXX, the message is scrutinized for destination address, parity error, and sync type. If the message is addressed to another terminal or if a parity error is detected, the interrupt routine is terminated. If the message is a data message, the program jumps to RECMSG. Command messages are handled by using the lower three bits of the data byte to perform an indirect jump to one of eight possible command handling routines.

Figure 44: Serial Message Decoder Routine Flow Chart

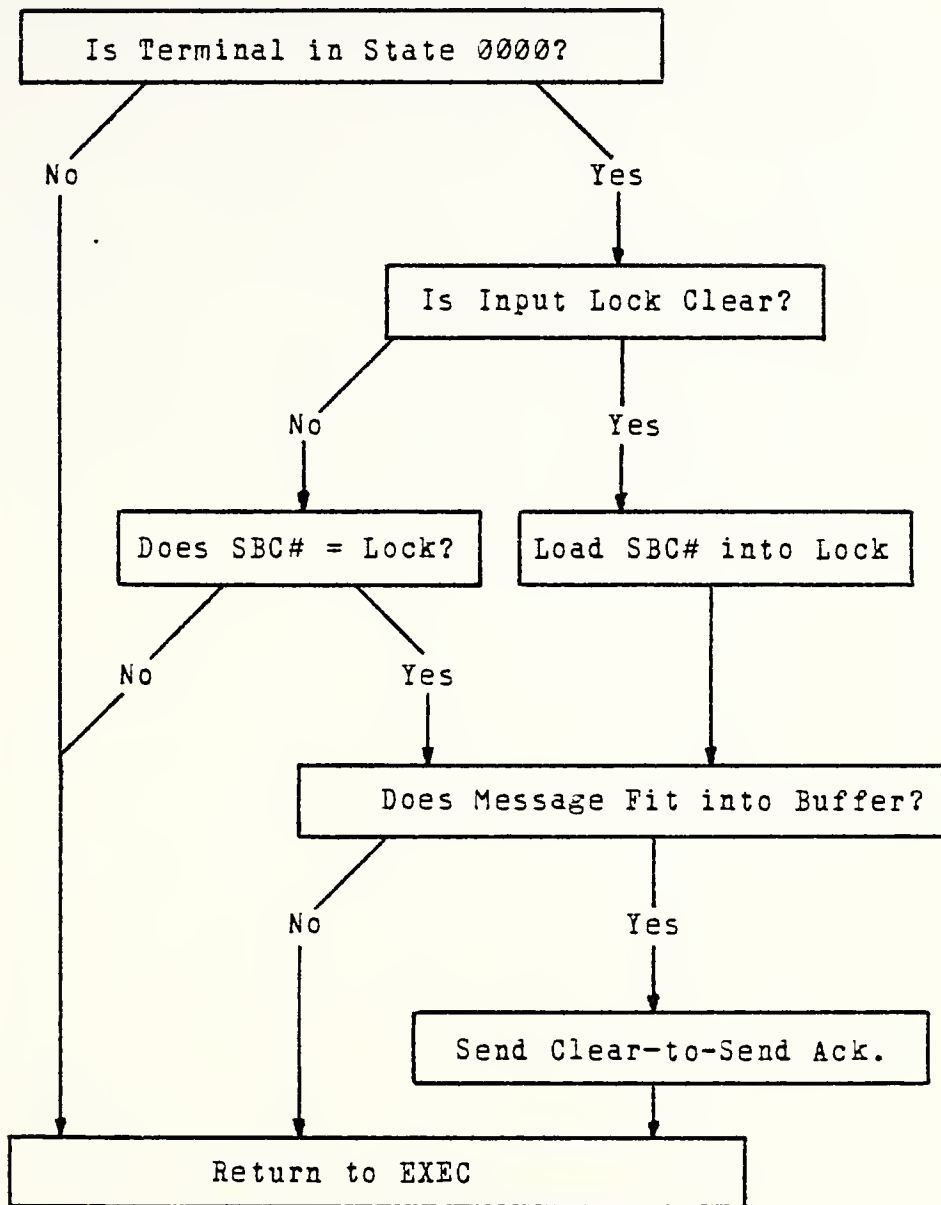


J. REQUEST-TO-SEND COMMAND SERVICE ROUTINE - REQCMD

This routine processes the request-to-send command and issues a clear-to-send acknowledgement if room is available for the message. If the terminal is not in state 0000, the command is rejected and the interrupt sequence terminated. If the input lock is clear, the requesting SBC number is loaded into the lock. If the lock is already set, the SBC number is compared against the lock to see if the lock was set by an earlier request from the same terminal. If the lock is already set and not equal to the requesting SBC number, a parallel output is in progress and the serial message can not be accommodated.

The high five bits of the request-to-send command carry all but the LSB of the message length. The message length is found by masking off the lower three command bits and shifting the remaining five bits to the right twice. This is then compared with the available room in the 8748 buffer. If sufficient room exists, the clear-to-send acknowledgement is issued. Otherwise, the routine ends without a transmission.

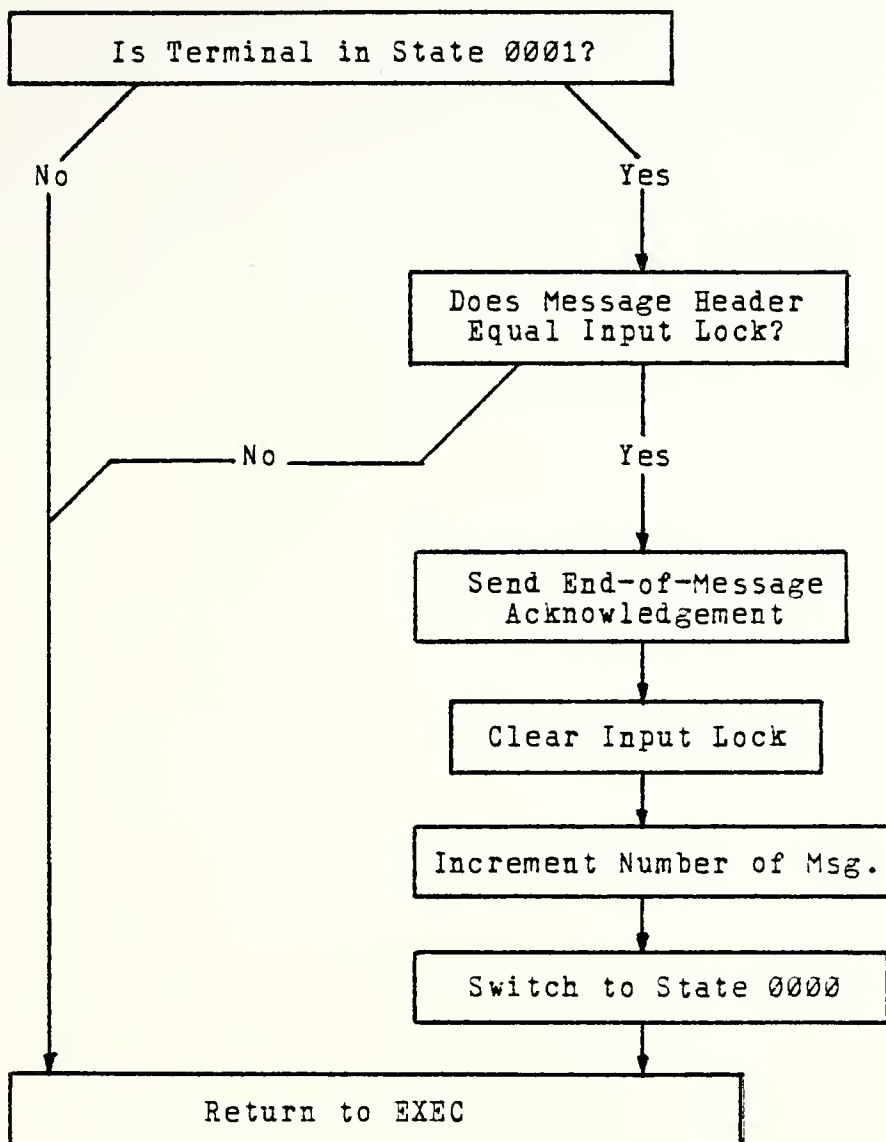
Figure 45: Request-to-Send Command Service Routine Flow Chart



K. END-OF-MESSAGE COMMAND SERVICE ROUTINE - ENDCMD

This routine is called from MSGDEC and services an end-of-message command. The message validity is tested by checking the terminal state for 0001 and comparing the SBC number of the sender against the input lock. Discrepancies cause the routine to be ended without changes to the 8748. If no further errors are detected, the 8748 input lock is cleared, the number of messages incremented, and the terminal state returned to 0000. A message is sent to the previous terminal to serve as an end-of-message acknowledgement.

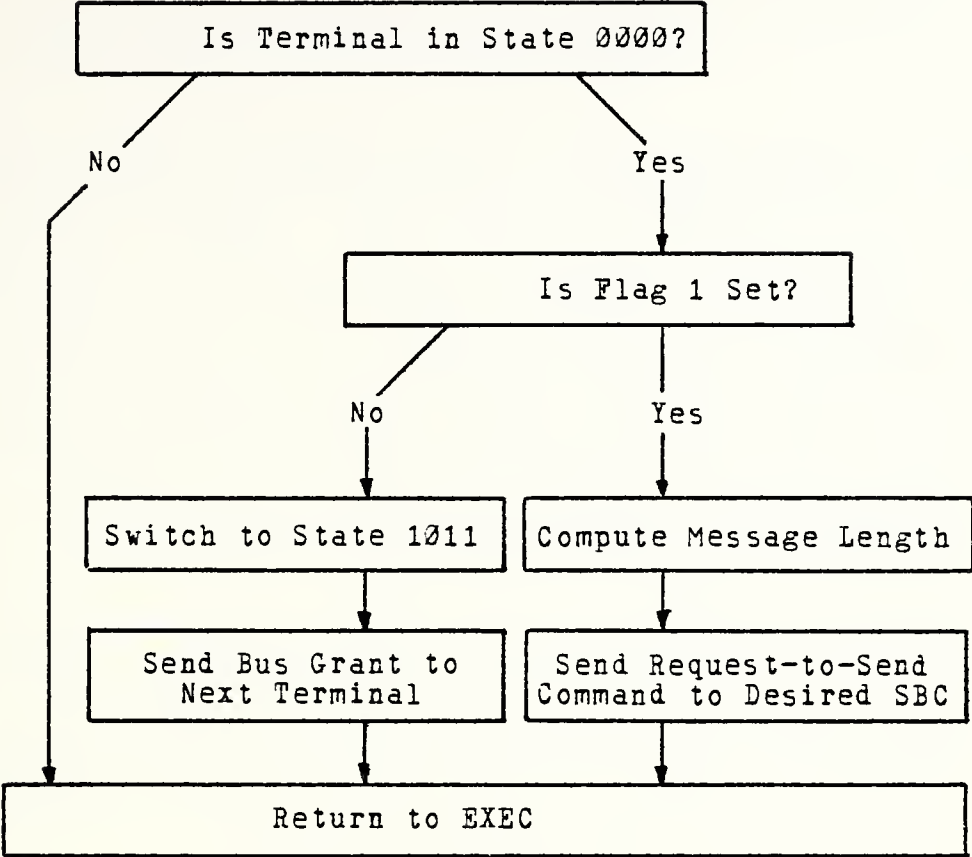
Figure 46: End-of-Message Command Service Routine Flow Chart



L. BUS GRANT COMMAND SERVICE ROUTINE - BUSCMD

This routine handles the bus grant command. The terminal must be in state 0000 to accept the bus grant command. Internal flag 0 indicates a message is waiting to be sent. If the flag is not set, the terminal switches to mode 1011 and issues a bus grant to the next station. If a message is waiting to be sent, the terminal computes the message length and issues a request-to-send command to the desired terminal.

Figure 47: Bus Grant Command Service Routine Flow Chart



M. ACKNOWLEDGEMENT ROUTINES: CLRACK, DATAACK, ENDACK, BUSACK

CLRACK handles a clear-to-send acknowledgement by switching the terminal state to 1001 and jumping to the DATAACK routine to send the first data message.

DATAACK checks for the end of the message before issuing the next data message. If no more bytes remain to be sent, the terminal state is switched to 1010 and an end-of-message command is issued. If more data is waiting to be sent, the next byte is fetched from the 8748 buffer and sent to the previous terminal via the SENDDP routine. The 8748 output pointer is checked for overflow and reset if necessary.

ENDACK clears the output lock and decrements the number of messages in the 8748. The terminal mode is switched to 1011 and a bus grant command is issued to the next terminal by SENDCN.

BUSACK switches the terminal mode to 0000 and returns to MSGDEC to decode the full message. This is done to handle a two terminal bus where bus commands are exchanged continuously between the two terminals. In this case the bus grant acknowledgement consists of a bus grant command and needs to be decoded as a command.

Figure 48: Clear-to-Send Acknowledgement Routine Flow Chart

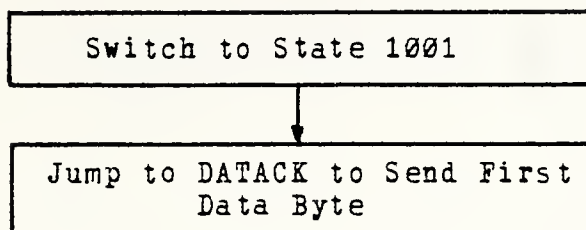


Figure 49: End-of-Message Acknowledgement Routine Flow Chart

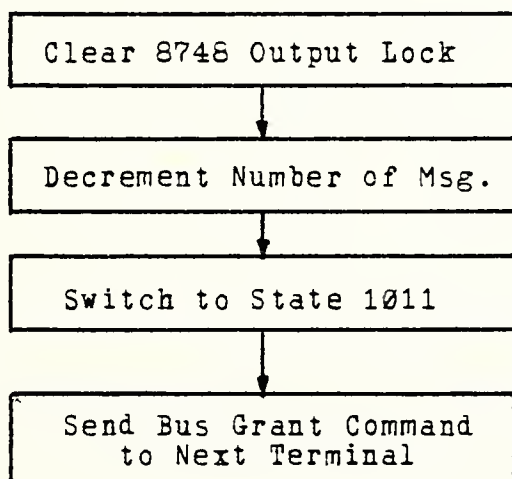


Figure 50: Bus Grant Acknowledgement Routine Flow Chart

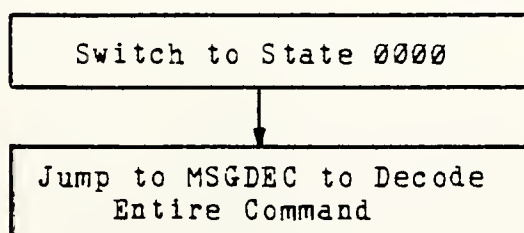
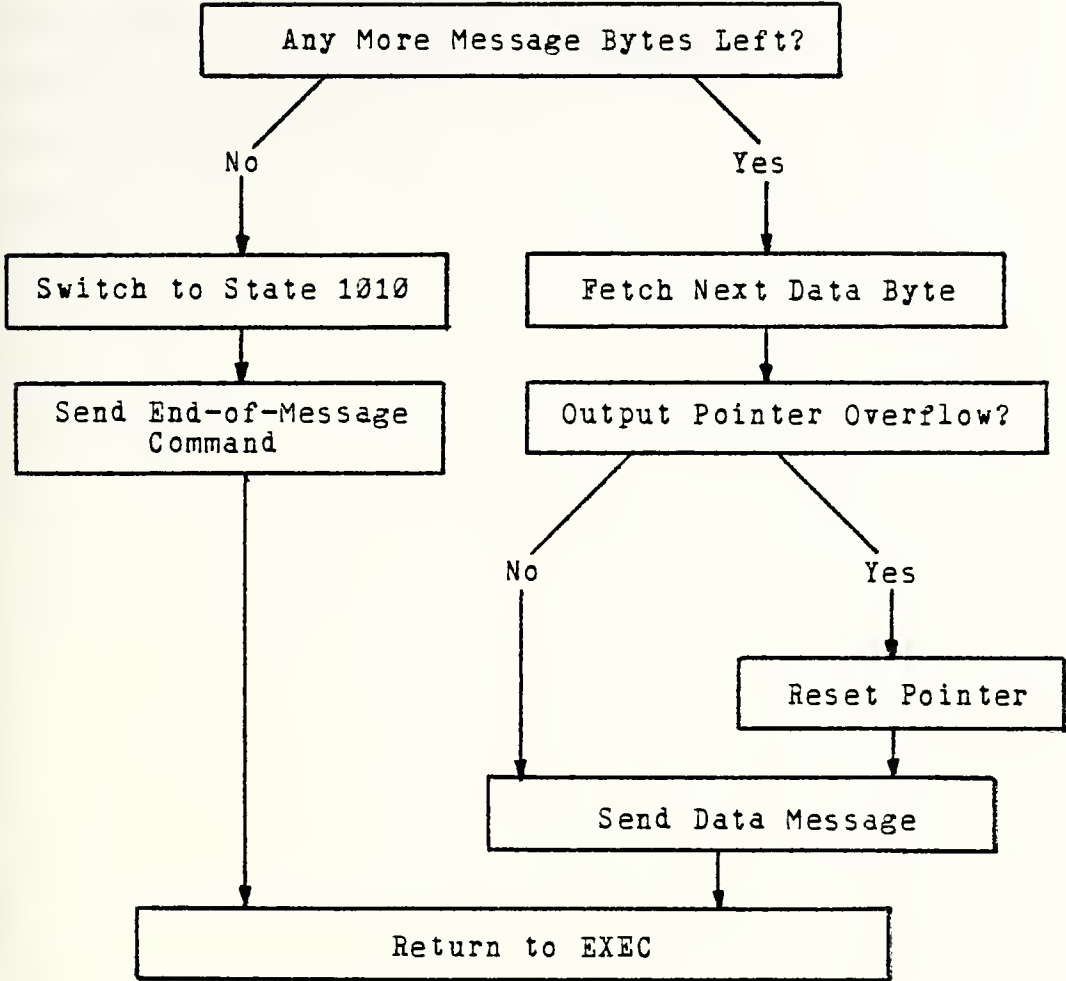


Figure 51: Data Message Acknowledgement Routine Flow Chart



N. INTERRUPT RETURN ROUTINE - RETURN

This routine cleans up after each time out or received message generated interrupt. The time out counter is reloaded and the accumulator is retrieved from memory. The routine may be entered in one of two places: at the beginning to reload the error counter, or after the reload instruction to leave the error counter unchanged. The latter case is used only on return from a time out interrupt when the number of consecutive time outs is accumulated. Received message interrupts reset the error counter. The interrupt routine is terminated by the RETR, return and restore routine.

Figure 52: Interrupt Return Routine Flow Chart

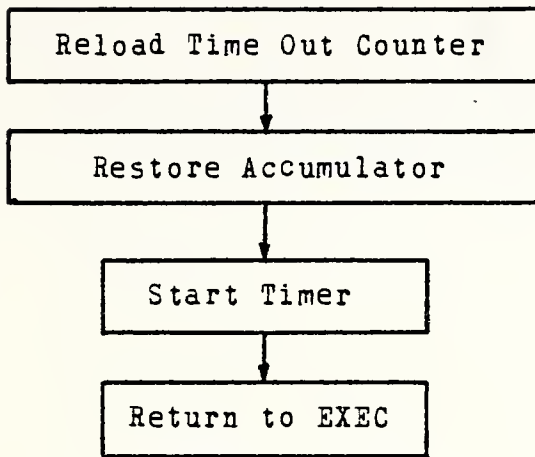


Table 2: Serial Bus Terminal States

State	Terminal Function
0000	Waiting for a serial message
0001	In process of receiving a message
0010	Not used
0011	Not used
0100	Not used
0101	Not used
0110	Not used
0111	Not used
1000	Bus controller requesting transfer
1001	Sending data message
1010	Ending a serial transfer
1011	Relinquishing bus control
1100	Not used
1101	Not used
1110	Not used
1111	Not used

Table 3: Serial Bus Command Codes

Command Code	Command
000	Request-to-send
001	End-of-message command
010	Bus grant command
011	Not used
100	Not used
101	Not used
110	Not used
111	Not used

Table 4: Corporate Memory Buffer and Control Registers

Address	Mnemonic	Function
F01C	EXTMSGLOCK	External buffer lock mechanism: =0 if butter is unlocked else = SBC# of current user
F01D	EXTMSG	Indicates a message is waiting =0 if buffer is empty; else = SBC# to receive next message
F01E	EXTMSGIN	Input pointer for incoming messages points to next address to be filled; pointer is rela- tive to bottom of buffer
F01F	EXTMSGOUT	Output pointer for outgoing messages similar to EXTMSGIN
F020	NUMEXTMSG	Number of messages in the buffer
F021	EXTMSGLAST	Indicates an outgoing message is the last one on the top of the stack by setting EXTMS- GLAST = EXTMSGOUT
F022 - F119	EXTMSGBUFFER	248 bytes of message buffer

Table 5: CPU Control Lines

Port #1 Control Bits

Bit	Function	Polarity
7	Msg Format 1	Normally Low
6	Msg Format 0	Normally Low
5	Parity Select	1=Even, 0=Odd Parity
4	Sync Type (Out)	1=Cmd, 0=Data
3	Send Request	Active Low
2	Receiver Ack	Active High
1	Read/Write	1=Read, 0=Write
0	Bus Request	Active Low

Port #2 Control Bits

Bit	Function	Polarity
7	Parity Error	Active High
6	Sync Type (In)	1=Data, 0=Cmd
5	SW 4	
4	SW 3	
3	SW 2	
2	SW 1	
1	Addr Bit 9	Used For Single-
0	Addr Bit 8	Step Diagnostics

Test Inputs

Bit	Function	Polarity
T1	Par. Bus Ack	Active High
T0	Send Ack	Active High

Table 6: CPU Latch Controls

Addr	Read Strobe	Write Strobe
X0	Read Serial Byte 0	Load Serial Byte 0
X1	Read Serial Byte 1	Load Serial Byte 1
X2	Read Serial Byte 2	Load Serial Byte 2
X3	Read Serial Byte 3	Load Serial Byte 3
X4	Read Parallel Data	Load Parallel Data
X5	" " "	Not Used**
X6	" " "	Load Parallel Addr H
X7	" " "	Load Parallel Addr L
X8	Not Used	Not Used
X9	"	"
XA	"	"
XB	"	"
XC	"	"
XD	"	"
XE	"	"
XF	"	"

** Reserved for 16 bit data bus expansion

Table 7: 8748 Memory Allocation

3F	32X8		Message Buffer
20			
1F	R7'		Output Counter
1E	R6'		Output Pointer
1D	R5'		Input Counter
1C	R4'		Input Pointer
1B	R3'		Terminal Mode
1A	R2'		Working Reg
19	R1'		Working Reg
18	R0'		Pointer
17			Bus Roster
16			Bus Format
15	Next	Own	ID Register
14	Dest	Src.	Output Lock
13	Dest	Src.	Input Lock
12			Numb of Msg
11			
10			
0F			
0E			
0D			5 Deep Stack
0C			
0B			
0A			
09			
08			
07	R7		High Addr Byte
06	R6		Low Addr Byte
05	R5		Working Reg
04	R4		" "
03	R3		" "
02	R2		" "
01	R1		" "
00	R0		Gen Pointer

APPENDIX C
PROGRAM LISTING

ROUTINE DIRECTORY

NAME	LENGTH	START ADDR	END ADDR
TIMOUT	34	007	027
RESTR	64	029	068
EXEC	123	069	0E5
RECEXT	144	0E6	175
SENDEXT	150	180	215
UNLOCK	9	217	21F
SETLOK	30	220	23D
SETEXT	25	240	258
WRITE	20	274	287
READ	19	288	29A
SENDDO	7	29D	2A3
SENDCO	7	2A4	2AA
SENDCI	8	2AB	2B2
SENDCN	16	2B3	2C2
RECMSG	30	2C4	2E1
MSGDEC	75	2E6	32F
REQCMD	50	339	36A
ENDCMD	27	36D	387
BUSCMD	28	389	3A4
CLRACK	4	3A9	3AC
DATAACK	28	3AE	3C9
ENDACK	18	3CA	3DB
BUSACK	4	3DC	3DF
RETURN	12	3E0	3EB
TOTAL	932		

TIMOUT	007	65	STOP TIMER	TIMER INTERRUPT
	008	D5	SEL RB1	
	009	B8 10	MOV R0',#10	PUSH A
	00B	A0	MOV GR0',A	
	00C	FB	MOV A,R3'	GET MODE
	00D	03 F8	ADD A,#F8	TEST FOR MODE=1000
	00F	96 19	JNZ TOUT1	
	011	BB 0B	MOV R3,#0B	SWITCH TO MODE 1011
	013	B9 02	MOV R1',#02	BUS GRANT COMMAND
	015	54 B3	CALL SENDCN	SEND TO NEXT TERM
	017	64 E0	JMP RETURN	WITH RELOAD
TOUT1	019	18	INC R0'	
	01A	F0	MOV A,GR0'	GET T/O COUNTER
	01B	07	DCR A	
	01C	A0	MOV GR0',A	RESTORE
	01D	C6 00	JZ RESTART	
	01F	FB	MOV A,R3'	GET MODE AGAIN
	020	37	CPL A	
	021	72 27	JB3 TOUT2	JUMP IF MODE=0XXX
	023	99 F7	ANL P1#F7	REPEAT LAST MSG
	025	89 08	ORL P1#08	
TOUT2	027	64 E4	JUMP RETURN	WITHOUT RELOAD

RESET	000	04 29	JMP RESTRT	POWER UP RESET
RESTRT	029	15	DIS I	
	02A	35	DIS TCNTI	
	02B	23 00	MOV A, #00	ZERO STACK PTR
	02D	D7	MOV PSW, A	
	02E	B9 0E	MOV R1, #0E	CLEAR REGISTERS
	030	B8 12	MOV R0, #12	12 THROUGH 1F
RST1	032	B0 00	MOV GR0, #00	
	034	18	INC R0	
	035	E9 32	DJNZ R1 RST1	
	037	23 A4	MOV A, #A4	RELOAD TIMER
	039	62	MOV T, A	
	03A	B8 11	MOV R0, #11	RELOAD ERROR CNTR
	03C	B0 0F	MOV GR0, #0F	
	03E	D5	SEL RB1	
	03F	BC 20	MOV R4', #20	INIT POINTERS
	041	BE 20	MOV R6', #20	
	043	C5	SEL RB0	
REREAD	044	BF F0	MOV R7, #F0	SET UP ADDR FOR BUS
	046	BE 00	MOV R6, #00	STATUS BYTE
	048	54 88	CALL READ	READ FORMAT
	04A	43 1B	ORL A, #1B	OUTPUT TO
	04C	39	OUTL P1, A	CONTROL PORT
	04D	1E	INC R6	
	04E	54 88	CALL READ	READ ID BYTE F001
	050	B8 15	MOV R0, #15	
	052	A0	MOV GR0, A	STORE IN R15

053	1E	INC R6	
054	54 88	CALL READ	READ ROSTER
056	C6 44	JZ REREAD	IF NO SBC'S
058	B8 17	MOV R0,#17	STORE IN R17
05A	A0	MOV GR0,A	
05B	D5	SEL RB1	
05C	37	CPL A	
05D	12 65	JB0	IF SBC#1 NOT LOCAL
05F	BB 0D	MOV R3',#0D	1011 BUS GRANT MODE
061	B9 02	MOV R1',#02	BUS GRANT COMMAND
063	54 B3	CALL SENDCN	SEND BUS GRANT
065	55	START TIMER	
066	25	EN TCNTI	
067	04 69	JMP EXEC	

OUTCHK	069	00	NOP	
	06A	00	NOP	
	06B	05	EN I	INTERRUPT WINDOW
	06C	00	NOP	
	06D	15	DIS I	
	06E	D5	SEL RB1	
	06F	B8 14	MOV R0',#14	GET OUTPUT LOCK
	071	F0	MOV A,GR0'	
	072	96 B6	JNZ INPCHK	IF OUTPUT IN PROG
	074	B8 12	MOV R0',#12	CHECK NUMB OF MSG
	076	F0	MOV A,GR0'	
	077	C6 B6	JZ INPCHK	NO MSG IN BUFFER
	079	FE	MOV A,R6'	GET RMN FROM NEXT
	07A	A8	MOV R0',A	OUTGOING MESSAGE
	07B	F0	MOV A,GR0'	
	07C	77	RR A	FORM SBC# FROM
	07D	77	RR A	MODULE NUMBER
	07E	77	RR A	
	07F	17	INC A	
	080	53 0F	ANL A,#0F	SBC# NOW IN A
	082	47	SWAP A	
	083	A9	MOV R1',A	
	084	18	INC R0'	
	085	F8	MOV A,R0'	
	086	37	CPL A	
	087	D2 8B	JB6	
	089	B8 20	MOV R0',#20	

08B	F0	MOV A,GR0'	
08C	77	RR A	
08D	77	RR A	
08E	77	RR A	
08F	17	INC A	
090	53 0F	ANL A,#0F	
092	69	ADD A,R1'	
093	B8 14	MOV R0',#14	LOAD SBC# INTO
095	A0	MOV GR0',A	OUTPUT LOCK
096	47	SWAP A	
097	43 F0	ORL A,#F0	FORM LOOK UP ADDR
099	E3	MOVP3 A,GA	GET SBC MASK
09A	A9	MOV R1',A	TEMP STORE MASK
09B	B8 17	MOV R0',#17	GET STATUS BITS
09D	F0	MOV A,GR0'	
09E	59	ANL A,R1'	SEE IF SBC IS LOCAL
09F	A5	CLR F1	
0A0	96 A3	JNZ EXEC1	IF LOCAL
0A2	B5	CPL F1	
EXEC1	0A3	FE	MOV A,R6'
			GET MSG LENGTH
0A4	03 03	ADD A,#03	
0A6	A8	MOV R0',A	
0A7	F0	MOV A,GR0'	PUT LENGTH IN
0A8	AF	MOV R7',A	OUTPUT COUNTER
0A9	C5	SEL RB0	
0AA	76 B6	JF1 INCHK	IF NOT LOCAL
0AC	00	NOP	

	0AD	00	NOP	
	0AE	54 24	CALL SETLOK	
	0B0	34 80	CALL SENDEX	
	0B2	54 17	CALL UNLOCK	
	0B4	04 69	JMP OUTCHK	
INCHK	0B6	05	EN I	DISABLE INTERRUPTS
	0B7	00	NOP	
	0B8	15	DIS I	
	0B9	D5	SEL RB1	
	0BA	B8 13	MOV R0',#13	READ INPUT LOCK
	0BC	F0	MOV A,@R0'	
	0BD	96 69	JNZ OUTCHK	SERIAL INPUT IN PROG
	0BF	C5	SEL RB0	
	0C0	BF F0	MOV R7,#F0	READ NUMB MSGS
	0C2	BE 20	MOV R6,#20	FROM COMMON MEM
	0C4	54 88	CALL READ	
	0C6	C6 69	JZ OUTCHK	
	0C8	BE 1D	MOV R6,#1D	
	0CA	54 88	CALL READ	READ EXTMSG FOR SBC#
	0CC	D5	SEL RB1	
	0CD	A9	MOV R1',A	TEMP STORE SBC#
	0CE	43 F0	ORL A,#F0	GET SBC MASK
	0D0	E3	MOVP3 A,@A	
	0D1	AA	MOV R2',A	
	0D2	B8 17	MOV R0',#17	GET STATUS BITS
	0D4	F0	MOV A,@R0'	
	0D5	5A	ANL A,R2'	TEST FOR SBC LOCAL

0D6	96 69	JNZ OUTCHK	LOCAL MESSAGE
0D8	B8 13	MOV R0',#13	IF NOT LOCAL,LOAD
0DA	F9	MOV A,R1'	INPUT LOCK WITH SBC#
0DB	A0	MOV GR0',A	
0DC	C5	SEL RB0	
0DD	00	NOP	
0DE	54 20	CALL SETLOK	INPUT
0E0	14 E6	CALL RECEXT	
0E2	54 17	CALL UNLOCK	
0E4	04 69	JMP OUTCHK	GO BACK FOR MORE

RECEXT	0E6	BF F0	MOV R7,#F0	READ OUTPUT
	0E8	BE 1F	MOV R6,#1F	POINTER
	0EA	54 88	CALL READ	
	0EC	AD	MOV R5,A	STORE IN R5
	0ED	1E	INC R6	
	0EE	54 88	CALL READ	GET NUMB OF MSG
	0F0	AB	MOV R3,A	IN R3
	0F1	1E	INC R6	
	0F2	54 88	CALL READ	GET EXTMSG LAST
	0F4	AC	MOV R4,A	IN R4
	0F5	FD	MOV A,R5	IS OUT = LAST ?
	0F6	37	CPL A	
	0F7	17	INC A	
	0F8	6C	ADD A,R4	
	0F9	96 FF	JNZ RECX1	OUT = LAST
	0FB	BD 00	MOV R5,#00	SET OUT=00
	0FD	BC 00	MOV R4,#00	SET LAST=00
RECX1	0FF	23 25	MOV A,#25	COMPUTE ADDR OF
	101	6D	ADD A,R5	MESSAGE LENGTH
	102	E6 05	JNC RECX2	
	104	1F	INC R7	BUMP R7 IF CARRY
RECX2	105	AE	MOV R6,A	GET LENGTH OF MSG
	106	54 88	CALL READ	
	108	AA	MOV R2,A	IN R2
	109	D5	SEL RB1	COMPUTE ROOM IN 8748
	10A	FC	MOV A,R4	GET INPUT POINTER
	10B	37	CPL A	

	10C	17	INC A	
	10D	6E	ADD A,R6'	COMPUTE OUT-IN=A
	10E	F6 14	JC RECX3	IF OUT>=IN
	110	03 20	ADD A,#20	IF OUT<IN
	112	24 1C	JMP RECX4	
RECX3	114	96 1C	JNZ RECX4	
	116	B8 12	MOV R0',#12	GET NUMB OF MSG'S
	118	F0	MOV A,GR0'	
	119	C6 10	JZ 110	
	11B	27	CLR A	
RECX4	11C	C5	SEL RB0	
	11D	37	CPL A	
	11E	00	NOP	
	11F	6A	ADD A,R2	IS LENGTH<ROOM?
	120	E6 31	JNC RECX5	IT FITS
	122	BF F0	MOV R7,#F0	DOESNT FIT
	124	BE 1F	MOV R6,#1F	
	126	FD	MOV A,R5	
	127	54 74	CALL WRITE	RESTORE EXTMSGOUT
	129	BE 21	MOV R6,#21	
	12B	FC	MOV A,R4	
	12C	54 74	CALL WRITE	RESTORE EXTMSGLAST
	12E	24 71	JMP RECX 11	GO BACK TO EXEC
	130	00	NOP	& CLEAR INPUT LOCK
RECX5	131	BF F0	MOV R7,#F0	BEGIN TRANSFER
	133	23 22	MOV A,#22	COMPUTE STARTING
	135	6D	ADD A,R5	ADDR OF FIRST BYTE

	136	AE	MOV R6,A	
	137	E6 3A	JNC RECX6	BUMP R7 IF CARRY
	139	1F	INC R7	
RECX6	13A	B8 1C	MOV R0,#1C	GET 8748 IN POINTER
	13C	F0	MOV A,0R0	IN R1
	13D	A9	MOV R1,A	
RECX7	13E	54 88	CALL READ	GET DATA BYTE
	140	A1	MOV 0R1,A	BUT IN 8748 BUFFER
	141	19	INC R1	BUMP 8748 IN POINTER
	142	23 C0	MOV A,#C0	CHECK IF POINTER>=64
	144	69	ADD A,R1	
	145	E6 49	JNC RECX8	
	147	B9 20	MOV R1,#20	IF >=64, POINTER=32
RECX8	149	23 01	MOV A,#01	INCR R6 ADDR POINTER
	14B	6E	ADD A,R6	
	14C	AE	MOV R6,A	
	14D	E6 50	JNC RECX9	IF R6 OVF, BUMP R7
	14F	1F	INC R7	
RECX9	150	1D	INC R5	BUMP RELATIVE PNTR.
	151	EA 3E	DJNZ R2 RECX7	GO BACK FOR MORE
	153	23 09	MOV A,#09	BEGIN CLEAN UP
	155	6D	ADD A,R5	IF REL POINTER OUT-
	156	E6 5A	JNC RECX10	SIDE BUFFER
	158	BD 00	MOV R5,#00	POINTER = 0
RECX10	15A	CB	DEC R3	DECR NUMB OF MSG
	15B	BF F0	MOV R7,#F0	IN COMMOM MEM
	15D	BE 1F	MOV R6,#1F	RESTORE POINTERS

15F	FD	MOV A,R5	
160	54 74	CALL WRITE	RESTORE EXTMSGOUT
162	1E	INC R6	
163	FB	MOV A,R3	
164	54 74	CALL WRITE	RESTORE EXTNUMBMSG
166	54 40	CALL SETEXT	FIND NEXT SBC #
168	B8 12	MOV R0,#12	INCR NUMB MSG 8748
16A	F0	MOV A,GR0	
16B	17	INC A	
16C	A0	MOV GR0,A	
16D	B8 1C	MOV R0,#1C	UPDATE 8748 INPUT
16F	F9	MOV A,R1	POINTER
170	A0	MOV GR0,A	
RECX11 171	B8 13	MOV R0,#13	CLR 8748 INPUT LOCK
173	B0 00	MOV GR0,00	
175	93	RETR	GO BACK TO EXEC

SENDEXT	180	00	NOP	
	181	BF F0	MOV R7,#F0	
	183	BE 1E	MOV R6,#1E	
	185	54 88	CALL READ	READ EXTMSG IN
	187	AC	MOV R4,A	STORE IN R4
	188	1E	INC R6	
	189	54 88	CALL READ	READ EXTMSG OUT
	18B	AD	MOV R5,A	STORE IN R5
	18C	1E	INC R6	
	18D	54 88	CALL READ	READ NUMEXTMSG
	18F	AB	MOV R3,A	STORE IN R3
	190	B8 1F	MOV R0,#1F	GET MSG LENGTH FROM
	192	F0	MOV A,GR0	8748 OUTPUT COUNTER
	193	AA	MOV R2,A	STORE IN R2
	194	FD	MOV A,R5	
	195	37	CPL A	
	196	17	INC A	COMPUTE IN-OUT
	197	6C	ADD A,R4	CY=1 IF IN>=OUT
	198	A9	MOV R1,A	TEMP STORE
	199	FB	MOV A,R3	GET NUMB OF MSG'S
	19A	C6 A1	JZ SENX1	JUMP IF NUMB=0
	19C	F9	MOV A,R1	GET IN-OUT
	19D	C6 BE	JZ SENX3	JUMP IF IN=OUT
	19F	E6 B3	JNC SENX2	JUMP IF OUT>IN
SENX1	1A1	FC	MOV A,R4	GET INPUT POINTER
	1A2	37	CPL A	COMPUTE ROOM=
	1A3	03 F8	ADD A,#F8	SIZE-INPUT POINTER

	1A5	A9	MOV R1,A	TEMP STORE ROOM IN 1
	1A6	FA	MOV A,R2	GET LENGTH
	1A7	37	CPL A	
	1A8	17	INC A	COMPUTE ROOM-LENGTH
	1A9	69	ADD A,R1	CY=1 IF ROOM>=LENGTH
	1AA	F6 C9	JC SENX4	BEGIN TRANSFER
	1AC	BE 21	MOV R6,#21	
	1AE	FC	MOV A,R4	SET LAST=IN
	1AF	54 74	CALL WRITE	
	1B1	BC 00	MOV R4,#00	SET IN=0
SENX2	1B3	FC	MOV A,R4	
	1B4	37	CPL A	
	1B5	17	INC A	COMPUTE OUT-IN
	1B6	6D	ADD A,R5	
	1B7	A9	MOV R1,A	TEMP STORE IN R1
	1B8	FA	MOV A,R2	
	1B9	37	CPL A	
	1BA	17	INC A	COMPUTE
	1BB	69	ADD A,R1	(OUT-IN)-LENGTH
	1BC	F6 C9	JC SENX4	BEGIN TRANSFER
SENX3	1BE	BE 1E	MOV R6,#1E	BUFFER IS FULL
	1C0	FC	MOV A,R4	RESTORE EXTMSGIN
	1C1	54 74	CALL WRITE	
	1C3	44 11	JMP SENX 11	RETURN TO EXEC
	1C5	00	NOP	& CLEAR OUTPUT LOCK
	1C6	00	NOP	
	1C7	00	NOP	

	1C8	00	NOP	
SENX4	1C9	BF F0	MOV R7,#F0	COMPUTE STARTING
	1CB	23 22	MOV A,#22	ADDRESS OF FIRST
	1CD	6C	ADD A,R4	BYTE TO BE SENT
	1CE	AE	MOV R6,A	IN R6
	1CF	E6 D2	JNC SENX5	
	1D1	1F	INC R7	BUMP R7 IF R6 OVF
SENX5	1D2	B8 1E	MOV R0,#1E	
	1D4	F0	MOV A,@R0	GET 8748 OUTPUT
	1D5	A9	MOV R1,A	POINTER IN R1
	1D6	F1	MOV A,@R1	GET RMN IN R5
	1D7	AD	MOV R5,A	FOR SETEXT
SENX6	1D8	F1	MOV A,@R1	GET NEXT DATA BYTE
	1D9	54 74	CALL WRITE	SEND IT
	1DB	19	INC R1	INC 8748 POINTER
	1DC	23 C0	MOV A,#C0	IS POINTER>=64?
	1DE	69	ADD A,R1	
	1DF	E6 E3	JNC SENX7	
	1E1	B9 20	MOV R1,#20	IF SO,POINTER=32
SENX7	1E3	23 01	MOV A,#01	
	1E5	6E	ADD A,R6	INCR LOW ADDR BYTE
	1E6	AE	MOV R6,A	
	1E7	E6 EA	JNC SENX8	
	1E9	1F	INC R7	BUMP HIGH BYTE
SENX8	1EA	1C	INC R4	INC RELATIVE POINTER
	1EB	EA D8	DJNZ R2 SENX6	GO BACK FOR MORE
	1ED	23 09	MOV A,#09	BEGIN CLEAN UP

	1EF	6C	ADD A,R4	IS REL. POINTER>=F7
	1F0	E6 F4	JNC SENX9	
	1F2	BC 00	MOV R4,#00	IF SO,POINTER=0
SENX9	1F4	1B	INC R3	INCR NUMB OF MSG
	1F5	BF F0	MOV R7,#F0	
	1F7	BE 1E	MOV R6,#1E	
	1F9	FC	MOV A,R4	
	1FA	54 74	CALL WRITE	RESTORE EXTMSGIN
	1FC	BE 20	MOV R6,#20	
	1FE	FB	MOV A,R3	
	1FF	54 74	CALL WRITE	RESTORE NUMEXTMSG
	201	FB	MOV A,R3	
	202	07	DEC A	
	203	96 08	JNZ SENX10	IS NUMEXTMSG=1?
	205	FD	MOV A,R5	IF SO THEN NEXTMSG=
	206	54 4C	CALL SETEXT	SBC OF MSG PUT IN
SENX10	208	B8 12	MOV R0,#12	
	20A	F0	MOV A,@R0	GET NUMB MSG 8748
	20B	07	DEC A	
	20C	A0	MOV @R0,A	DEC NUMB MSG IN 8748
	20D	B8 1E	MOV R0,#1E	UPDATE 8748 OUTPUT
	20F	F9	MOV A,R1	POINTER
	210	A0	MOV @R0,A	
SENX11	211	B8 14	MOV R0,#14	CLEAR 8748 OUTPUT
	213	B0 00	MOV @R0,#00	LOCK
	215	93	RETR	RETURN TO EXEC

UNLOCK	217	BF F0	MOV R7,#F0	
	219	BE 1C	MOV R6,#1C	
	21B	23 00	MOV A,#00	
	21D	54 74	CALL WRITE	LOAD 0 IN EXTMSGLOCK
	21F	93	RETR	RETURN

SETLKI	220	B8 13	MOV R0,#13	GET INPUT LOCK SBC#
	222	44 26	JMP SETL1	
SETLK0	224	B8 14	MOV R0,#14	GET OUTPUT LOCK SBC#
SETL1	226	F0	MOV A,GR0	GET LOCK IN R5
	227	AD	MOV R5,A	
	228	BF F0	MOV R7,#F0	
	22A	BE 1C	MOV R6,#1C	
SETL2	22C	05	EN I	INTERRUPT WINDOW
	22D	00	NOP	
	22E	15	DIS I	
	22F	54 88	CALL READ	READ EXTMSGLOCK
	231	96 2C	JNZ SETL2	TRY AGAIN TILL 0
	233	FD	MOV A,R5	
	234	54 74	CALL WRITE	WRITE SBC# INTO LOCK
	236	54 88	CALL READ	MAKE SURE YOU GOT IT
	238	37	CPL A	
	239	17	INC A	
	23A	6D	ADD A,R5	IS EXTMSGLOCK=LOCK
	23B	96 2C	JNZ SETL2	IF NOT, TRY AGAIN
	23D	93	RETR	RETURN IF LOCKED

SETEXT	240	FB	MOV A,R3	GET NUM OF MSG
	241	C6 52	JZ SETX2	JUMP IF NUMBER=0
	243	FD	MOV A,R5	GET NXT OUTGOING RMN
	244	03 22	ADD A,#22	FORM ADDRESS OF NEXT
	246	AE	MOV R6,A	MODULE NUMBER
	247	E6 4A	JNC SETX1	
	249	1F	INC R7	BUMP R7 IF R6 OVF
SETX1	24A	54 88	CALL READ	READ NEXT RMN
	24C	77	RR A	
	24D	77	RR A	DIVIDE RMN BY 8
	24E	77	RR A	
	24F	17	INC A	
	250	53 0F	ANL A,#0F	MASK OFF GARBAGE
SEXT2	252	BF F0	MOV R7,#F0	
	254	BE 1D	MOV R6,#1D	
	259	54 74	CALL WRITE	LOAD SBC INTO EXTMSG
	258	93	RETR	RETURN

WRITE	274	B8 04	MOV R0,04	SET UP LATCH ADDR
	276	37	CPL A	
	277	90	MOVX GR0,A	LOAD DATA BYTE
	278	18	INC R0	
	279	18	INC R0	FORM NEXT LATCH ADDR
	27A	FE	MOV A,R6	
	27B	37	CPL A	
	27C	90	MOVX GR0,A	LOAD LOW ADDR BYTE
	27D	18	INC R0	
	27E	FF	MOV A,R7	
	27F	37	CPL A	
	280	90	MOVX GR0,A	LOAD HIGH ADDR BYTE
	281	99 FC	ANL P1,#FC	ACTIVATE BUS REQ
WRT1	283	46 83	JNT1 WRT1	WAIT FOR XACK
	285	89 03	ORL P1,#03	REMOVE BUS REQ
	287	93	RETR	RETURN

READ	288	B8 07	MOV R0,#07	LATCH ADDRESS
	28A	FF	MOV A,R7	
	28B	37	CPL A	
	28C	90	MOVX GR0,A	LOAD HIGH ADDR BYTE
	28D	C8	DEC R0	
	28E	FE	MOV A,R6	
	28F	37	CPL A	
	290	90	MOVX GR0,A	LOAD LOW ADDR BYTE
	291	89 02	ORL P1,#02	RESET WRITE BIT
	293	99 FE	ANL P1,#FE	ACTIVATE BUS REQ
READ1	295	46 95	JNT1 READ1	WAIT FOR XACK
	297	80	MOVX A,GR0	READ BUS DATA
	298	89 01	ORL P1,#01	RESET BUS REQUEST
	29A	93	RETR	RETURN

SENDDO	29D	99 EF	ANL P1 EF	DATA SYNC GET OUTPUT
	29F	B8 14	MOV R0',14	LOCK IN A
	2A1	F0	MOV A,@R0'	
	2A2	44 B4	JMP SEND	
SENDCO	2A4	89 10	ORL P1,#10	COMMAND SYNC
	2A6	B8 13	MOV R0',#14	GET OUTPUT LOCK
	2A8	F0	MOV A,@R0	
	2A9	44 B8	JMP SEND	
SENDCI	2AB	89 10	ORL P1,#10	
	2AD	B8 13	MOV R0',#13	GET INPUT LOCK
	2AF	F0	MOV A,@R0	
	2B0	47	SWAP A	REVERSE THE ORDER
	2B1	44 B8	JMP SEND	OUTPUT IT
SENDCN	2B3	89 10	ORL P1,#10	SET COMMAND SYNC
	2B5	B8 15	MOV R0',#15	GET ID REGISTER
	2B7	F0	MOV A,@R0'	
SEND	2B8	B8 00	MOV R0',#00	LOAD HEADER
	2BA	90	MOVX @R0',A	
	2BB	18	INC R0'	
	2BC	F9	MOV A,R1'	GET DATA BYTE
	2BD	90	MOVX @R0',A	LOAD DATA
	2BE	99 F7	ANL P1,#F7	ACTIVATE SEND BIT
	2C0	89 08	ORL P1,#08	CLEAR SEND BIT
	2C2	83	RET	NO RESTORE

RECMSG	2C4	FB	MOV A,R3'	CHECK MODE=0001
	2C5	07	DEC A	
	2C6	96 E0	JNZ TO RETURN	
RECM1	2C8	B8 13	MOV R0',#13	GET INPUT LOCK
	2CA	F0	MOV A,@R0'	
	2CB	37	CPL A	
	2CC	17	INC A	
	2CD	6A	ADD A,R2'	COMPARE WITH SBC#
	2CE	96 E0	JNZ TO RETURN	
RECM2	2D0	FC	MOV A,R4'	GET INPUT POINTER
	2D1	A8	MOV R0',A	IN R0'
	2D2	F9	MOV A,R1'	GET DATA BYTE R1'
	2D3	A0	MOV @R0',A	STORE IN MEMORY
	2D4	1C	INC R4'	
	2D5	FC	MOV A,R4'	
	2D6	03 C0	ADD A,#C0	CHECK POINTER OVF
	2D8	E6 DC	JNC RECM3	
	2DA	BC 20	MOV R4',#20	
RECM3	2DC	B9 FF	MOV R1',#FF	
	2DE	54 AB	CALL SENDCI	
	2E0	64 E0	JMP RETURN	

EXINT	003	44 E8	JMP MSGDEC	
	2E6	64 16	JMP ACKSRV	
MSGDEC	2E8	89 04	ORL P1,#04	CLR MSG RDY FF
	2EA	99 FB	ANL P1,#FB	
	2EC	65	STOP TCOUNT	
	2ED	D5	SEL RB1	
	2EE	B8 10	MOV R0',#10	SAVE A
	2F0	A0	MOV GR0',A	
	2F1	FB	MOV A,R3'	GET MODE
	2F2	72 E6	JB3 ACKSRV	JUMP IF BUS CONTROL
	2F4	B8 01	MOV R0',#01	GET MSG HEADER IN R2
	2F6	80	MOVX A,GR0'	
	2F7	AA	MOV R2',A	
	2F8	47	SWAP A	GET "TO" ADDR BITS
	2F9	43 F0	ORL A,#F0	FORM LOOK UP ADDR
	2FB	E3	MOVP3 A,GA	GET STATUS BITS
	2FC	A9	MOV R1',A	TEMP STORE
	2FD	B8 17	MOV R0',#17	
	2FF	F0	MOV A,GR0'	COMP WITH TERMINAL
	300	59	ANL A,R1'	STATUS
	301	C6 E0	JZ RETURN	
	303	B8 00	MOV R0',#00	GET DATA BYTE IN R1'
	305	80	MOVX A,GR0'	
	306	A9	MOV R1',A	
	307	00	NOP	
	308	00 00	NOP	
	30A	00	NOP	

	30B	0A	IN A,P2	PARITY AND SYNC TYPE
	30C	D2 E0	JB6 RETURN	PARITY ERROR
	30E	F2 1C	JB7 RECMSG	DATA MESSAGE
	310	F9	MOV A,R1'	
	311	53 07	ANL A,#07	FORM JUMP TABLE ADDR
	313	03 20	ADD A,#20	ADD TABLE 1 OFFSET
	315	B3	JMPP QA	JUMP TO SERV ROUTINE
ACKSRV	316	F3	MOV A,R3'	GET MODE
	317	53 07	ANL A,#07	MASK LOW 3 BITS
	319	03 28	ADD A,#28	ADD TABLE 2 OFFSET
	31B	B3	JMPP QA	ACK SERVICE TABLE
MSGDC1	31C	44 C4	JMP RECMSG	
	320	39	REQCMD	LEAVES WITH DATA
	321	6D	ENDCMD	IN R1' HEADER IN R2
	322	89	BUSCMD	
	323	E0	RETURN	
	324	E0	RETURN	
	325	E0	RETURN	
	326	E0	RETURN	
	327	E0	RETURN	
	328	A9	CLRACK	
	329	AE	DATAACK	
	32A	CA	ENDACK	
	32B	DC	BUSACK	
	32C	DC	BUSACK	
	32D	DC	BUSACK	

32E DC BUSACK

32F DC BUSACK

REQCMD	339	FB	MOV A,R3'	CHECK FOR MODE 0000
	33A	96 E0	JNZ RETURN	REJECT IF MODE NOT 0
	33C	B8 13	MOV R0',#13	CHECK HEADER=IN LOCK
	33E	F0	MOV A,GR0'	
	33F	37	CPL A	
	340	17	INC A	
	341	6A	ADD A,R2'	
	342	C6 49	JZ REQC1	IF HEADER=INPUT LOCK
	344	F0	MOV A,GR0'	CHECK INPUT LOCK = 0
	345	96 E0	JNZ RETURN	NOT CLEAR
	347	FA	MOV A,R2'	MOVE HEADER TO LOCK
	348	A0	MOV GR0',A	
REQC1	349	C8	DEC R0'	
	34A	F0	MOV A,GR0'	GET NUMB OF MSG'S
	34B	96 50	JNZ REQC2	
	34D	27	CLR A	BUF. EMPTY ROOM=SIZE
	34E	64 56	JMP REQC3	
REQC2	350	FC	MOV A,R4'	COMPUTE OUT-IN
	351	37	CPL A	
	352	17	INC A	
	353	6E	ADD A,R6'	A=OUT-IN
	354	F6 56	JC REQC4	FITS
REQC3	356	03 40	ADD A,#40	A=SIZE-(IN-OUT)
REQC4	358	AA	MOV R2',A	TEMP STORE ROOM
	359	F9	MOV A,R1'	GET COMMAND BYTE
	35A	53 F8	ANL A,#F8	MASK OFF SIZE BITS
	35C	77	RR A	

35D	77	RR A	
35E	37	CPL A	
35F	17	INC A	IF ROOM>=LENGTH
360	6A	ADD A,R2'	CARRY = 1
361	E6 E0	JNC RETURN	NOT CLEAR
363	BB 01	MOV R3',#01	SET MODE 0001
365	B9 FF	MOV R1',#FF	
367	54 AB	CALL SENDCI	
369	64 E0	JMP RETURN	

ENDCMD	36D	23 FF	MOV A,#FF	CHECK FOR MODE 0001
	36F	6B	ADD A,R3'	
	370	96 E0	JNZ RETURN	IF MODE NOT 0001
	372	B8 13	MOV R0',#13	CHECK HEADER=IN LOCK
	374	F0	MOV A,GR0'	
	375	37	CPL A	
	376	17	INC A	
	377	6A	ADD A,R2'	
	378	96 E0	JNZ RETURN	IF WRONG SBC#
	37A	B9 FF	MOV R1',#FF	SEND ENDACK
	37C	54 AB	CALL SENDCI	
	37E	B0 00	MOV GR0',#00	CLEAR INPUT LOCK
	380	C8	DEC R0'	
	381	F0	MOV A,GR0'	INCR NUMB OF MGS
	382	17	INC A	
	383	A0	MOV GR0,A	
	384	BB 00	MOV R3',#00	SWITCH TO MODE 0000
	386	64 E0	JMP RETURN	

BUSCMD	389	FB	MOV A,R3'	CHECK FOR MODE 0000
	38A	C6 8E	JZ BUSC1	
	38C	64 E0	JMP RETURN	WITH RESET
BUSC1	38E	76 98	JF1 BUSC2	JMP IF MSG WAITING
	390	BB 0B	MOV R3',#0B	SWITCH TO MODE 1011
	392	B9 02	MOVE R1',#02	BUS GRANT CMD
	394	54 B3	CALL SENDCN	SEND BUS GRANT
	396	64 E0	JMP RETURN	
BUSC2	398	BB 08	MOV R3',08	SWITCH TO MODE 1000
	39A	FF	MOV A,R7'	GET MSG LENGTH
	39B	17	INC A	
	39C	53 3E	ANL A,#3E	COMPUTE LENGTH
	39E	E7	RL A	FOR SEND REQUEST
	39F	E7	RL A	
	3A0	A9	MOV R1',A	
	3A1	54 A4	CALL SENDCO	
	3A3	64 E0	JMP RETURN	

CLRACK	3A9	BB 09	MOV R3',#09	SWITCH MODE 1001
	3AB	64 B9	JMP DATK1	SEND FIRST DATA BYTE

DATAACK	3AE	FF	MOV A,R7'	ANY MORE BYTES LEFT?
	3AF	96 B9	JNZ DATK1	SEND ANOTHER
	3B1	BB 0A	MOV R3',#0A	SWITCH MODE 1010
	3B3	B9 01	MOV R1',#01	SEND END OF MSG
	3B5	54 A4	CALL SENDCO	
	3B7	64 E0	JMP RETURN	
DATK1	3B9	FE	MOV A,R6'	GET OUTPUT POINTER
	3BA	A8	MOV R0',A	
	3BB	F0	MOV A,R0'	GET NEXT DATA BYTE
	3BC	A9	MOV R1',A	IN R1'
	3BD	CF	DEC R7'	DECR OUTPUT COUNTER
	3BE	1E	INC R6'	BUMP THE POINTER
	3BF	FE	MOV A,R6'	IS POINTER AT TOP?
	3C0	03 C0	ADD A,#C0	
	3C2	E6 C6	JNC DATK2	
	3C4	BE 20	MOV R6',#20	POINTER AT BOTTOM
DATK2	3C6	54 9D	CALL SENDDO	SEND DATA
	3C8	64 E0	JMP RETURN	

ENDACK	3CA	B8 14	MOV R0',#14	CLR OUTPUT LOCK
	3CC	B0 00	MOV GR0',#00	
	3CE	B8 12	MOV R0',#12	DECR NUMB OF MSG'S
	3D0	F0	MOV A,GR0'	
	3D1	07	DCR A	
	3D2	A0	MOV GR0',A	
	3D3	A5	CLR F1	
	3D4	BB 0B	MOV R3',#0B	SWITCH MODE 1011
	3D6	B9 02	MOV R1',#02	BUS GRANT CMD
	3D8	54 B3	CALL SENDCN	TO NEXT TERMINAL
	3DA	64 E0	JMP RETURN	

BUSACK	3DC	BB 00	MOV R3',#00	SWITCH MODE 0000
	3DE	44 F4	JMP MSGDEC	

RETURN

RETURN	3E0	B8 10	MOV R0',#10	RET FROM MSG INTRPT
RETWC	3E2	B0 0F	MOV GR0',#0F	RELOAD T/O CNTR
RETNC	3E4	23 A4	MOV A,#A4	RELOAD TIMER
	3E6	62	MOV T,A	
	3E7	B8 10	MOV R0',#10	POP A
	3E9	F0	MOV A,GR0	
	3EA	55	STRT T	
	3EB	93	RETR	RETURN RESTORE

APPENDIX D

DEBUGGING TECHNIQUES

The Intel ICE-48 in circuit emulator is necessary for efficient debugging of new 8748 programs. The ICE-48 system replaces the 8748 chip with an interface cable to an Intel MDS microcomputer development system. The ICE-48 software in the MDS allows the MDS to step through the program and display the contents of the internal registers and memory.

At the time of the software development, the emulator system was not available. A substitute system was developed with an Intel SBC 80/20 to perform a similar task. The RAM loader and single stepper routines included in this section load the external program memory and step the 8748 through the program. Addresses are displayed on the CRT terminal connected to the 80/20. To display the contents of the internal memory, a separate routine to write the contents of the internal memory into external memory can be used. The external memory can then be displayed on the CRT by the SBC monitor.

Debugging the bus control routines requires at least two serial interface boards up and running at the same time. This was accomplished by using two SBC 80/20's to step two serial interface boards through their routines simultaneously. A storage oscilloscope was used to monitor bus traffic,

since serial transmissions occur at normal speed in the single step mode.

The basic debugging procedure is to load the program into both serial boards and step both terminals through the restart routines. If the corporate message buffer is empty, the two terminals will exchange bus grant commands indefinitely. A message can be simulated by loading data into the corporate memory, through the SBC monitor, and setting the buffer control registers accordingly. The message transfer process can be observed one step at a time with the messages displayed on the oscilloscope.

If required, the hardware can be debugged by manual methods. The receiver shift registers (74LS164's) can be removed from their sockets and replaced with a switch bank composed of DIP switches and pull up resistors. This enables received messages to be simulated without requiring a second operational terminal. The 74LS175 at B16 may be removed and a TTL level signal from a pulse generator may be applied through pin 10. By manually pulsing the signal generator, the transmitter and encoder can be stepped through the transmit sequence. Removing the 74LS86 at D18 and grounding pin 8 will disable the receive parity check. Command message reception can be simulated by removing the 74LS161 at A17 and grounding pin 12. Allowing this pin to float will indicate data sync to the CPU. To create a received message interrupt, momentarily ground pin 1 of D19.

A. RAM LOADER PROGRAM

This program runs on an Intel SBC 80/20-04 and loads 1024 bytes from an external memory board to the serial interface read/write program memory. In the present version, the memory locations D000 through D3FF (hex) correspond to 8746 program memory 0000 through 03FF. The program resides in external memory starting at address C700. Program and data are stored on floppy disc by use of CP/M DDT.

The RAM loader program is entered through the SBC monitor command "G0700 (CR)". If the serial interface board is not in the reset mode, the RAM load is aborted and the error symbol "#" is displayed on the CRT. After all 1024 locations have been loaded, the program returns control to the SBC monitor and the prompt symbol "." is displayed.

Port E5 and EA of the SBC 80/20 are used as bidirectional data/address ports. The bus drivers are removed and bypassed to connect the 8255's directly to the interface cable. Port E6 serves as the control port, supplying chip select, write enable, address latch and single step signals to the serial interface board. A 74LS00 is used as a driver for the port E6 signals, hence all lines are inverted. After initial power-up reset of the SBC, all output ports are set in the input mode, driving the control lines low. After the RAM loader has been exercised, port E6 is left in the output mode, keeping the control lines high and enabling normal operation of the serial hardware.

The interface cable connects to J1 on the SBC 80/20. Connections to port E5 are made by installing jumpers in sockets A5 and A6. Only the lower four bits of E5 are used and the 74LS00 is installed in socket A4. Connections to port EA are made via jumpers from socket A3 pins eight and nine to socket A10 pins one and four. The MSB of port E9 tests the state of the reset switch, and is connected to the interface cable by jumpering socket A12 pin 12 to socket A3 pin six.

RAM LOADER PROGRAM

0700	3E 83	MVI A,83	SET INPUT PORTS
0702	D3 EB	OUT EB	
0704	DB E9	IN E9	CHECK FOR RESET LOW
0706	E6 80	ANI 80	
0708	C2 3A 03	JNZ 033A	DISPLAY ERROR ON CRT
070B	3E 82	MVI A,82	CHANGE OUTPUT PORTS
070D	D3 EB	OUT EB	
070F	3E 80	MVI A,80	
0711	D3 E7	OUT E7	PORT B OUT, PORT C OUT
0713	3E 00	MVI A,00	CLEAR CONTROL PORT
0715	D3 E6	OUT E6	
0717	21 00 D0	LXI H,D000	START ADDR OF DATA
071A	7C	MOV A,H	
071B	D3 EA	OUT EA	OUTPUT HIGH BYTE
071D	7D	MOV A,L	
071E	D3 E5	OUT E5	OUTPUT LOW BYTE
0720	3E 04	MVI A,04	
0722	D3 E6	OUT E6	APPLY ALE PULSE
0724	3E 00	MVI A,00	
0726	D3 E6	OUT E6	REMOVE ALE PULSE
0728	7E	MOV A,M	PUT DATA ON PORT E5
0729	D3 E5	OUT E5	
072B	00	NOP	WAIT FOR RINGING TO SETTLE
072C	00	NOP	
072D	00	NOP	

072E	00	NOP	
072F	00	NOP	
0730	00	NOP	
0731	00	NOP	
0732	3E 03	MVI A,03	APPLY WE AND CS VIA PORT E6
0734	D3 E6	OUT E6	
0736	3E 00	MVI A,00	REMOVE WE AND CS
0738	D3 E6	OUT E6	
073A	23	INX H	COMPUTE NEXT ADDRESS
073B	3E D4	MVI A,D4	COMPARE WITH END OF FILE D400H
073D	BC	CMP H	
073E	C2 1A 07	JNZ 071A	GO BACK FOR NEXT DATA
0741	3E 82	MVI A,82	ALL DONE
0743	D3 E7	OUT E7	TURN OFF OUTPUT PORTS
0745	3E 83	MVI A,83	
0747	D3 EB	OUT EB	
0749	CD 12 03	CALL 0312	CR ROUTINE IN MONITOR
074C	C3 3C 00	JMP 003C	SBC MONITOR

B. SINGLE STEPPER ROUTINE

The single stepper routine uses the SBC 80/20 to step the 8748 through its program and to display the addresses on the CRT console. The routine is entered by typing the monitor command "G0760 (CR)". The SBC 80/20 applies a pulse to the single step flip flop via the interface cable and then reads the contents of the 8748 program counter on the eight bits of the bus port and the lower four bits of port 2. This address is displayed on the CRT. The process is repeated each time a key is pressed on the keyboard. Depressing the "break" key causes the program to cycle at a rapid rate.

The single stepper can reside either in read/write memory or in ROM. If the SBC 80/20 is installed in the same card cage as the serial interface and if the 80/20 program lies in external memory, the system will 'lock up' when the serial board is stepped through a bus read or write. For this reason the single stepper program should reside in on-board RAM or ROM.

To insure the 8748 begins at address 0000, the single step switch in the serial board switch register should be set before the reset switch is open. The single step program may be exited at any time by depressing the carriage return. Control is returned to the SBC monitor and the

prompt symbol "." is displayed. The single step routine may be reentered and single stepping resumed from the last address displayed.

SINGLE STEP PROGRAM

0760	3E 82	MVI A,82	PORT B & C INPUT
0762	D3 E7	OUT E7	PORT C E6 OUTPUT
0764	3E 83	MVI A,83	
0766	D3 EB	OUT EB	
0768	3E 00	MVI A,00	INIT PORT E6
076A	D3 E6	OUT E6	
076C	CD 12 03	CALL 0312	OUTPUT CR, LF
076F	16 00	MVI D,00	INIT LINE COUNTER
0771	C3 91 07	JMP 0791	READ INITIAL STATE
0774	CD 12 03	CALL 0312	
0777	16 00	MVI D,00	INIT LINE COUNTER
0779	CD F4 02	CALL 02F4	WAIT FOR PRESSED KEY
077C	FE 8D	CPI ED	
077E	CA 49 07	JZ 0749	IF SO, ALL DONE
0781	3E 08	MVI A,08	OUTPUT SNGSTP PULSE
0783	D3 E6	OUT E6	
0785	3E 00	MVI A,00	REMOVE SNGSTP PULSE
0787	D3 E6	OUT E6	
0789	00	NOP	WAIT FOR ADDR LINE
078A	00	NOP	TO STABILIZE
078B	00	NOP	
078C	00	NOP	
078D	00	NOP	
078E	00	NOP	
078F	00	NOP	

0790	00	NOP	
0791	DB EA	IN EA	READ HIGH ADDR BYTE
0793	E6 03	ANI 03	MASK OFF HIGH BITS
0795	CD 0D 05	CALL 050D	DISPLAY ON CRT
0798	4F	MOV C,A	
0799	CD 07 03	CALL 0307	
079C	DB E5	IN E5	READ LOW ADDR BYTE
079E	CD 87 04	CALL 0487	OUTPUT ON CRT
07A1	0E 20	MVI C,20	ASCII FOR "SPACE"
07A3	CD 07 03	CALL 0307	PRINT SPACE
07A6	14	INR D	INCREMENT LINE CNTR
07A7	7A	MOV A,D	CHECK F/END OF LINE
07A8	FE 10	CPI 10	IF LINE CNTR < 16
07AA	C2 79 07	JNZ 0779	GO BACK F/NEXT PULSE
07AD	C3 74 07	JMP 0774	IF FULL-JUMP CR-LF

C. 8748 REGISTER DUMP ROUTINE

This routine can be used to write the contents of the 8748 internal registers into external memory. The memory contents can then be examined using the SBC monitor. This routine may be located where space permits in the 8748 software. The routine is entered by inserting a CALL command into the 8748 EXEC routine. The memory contents are moved to address E000 (hex) through E03F in external memory.

```
C5      SEL RB0
BF E0   MOV R7,#E0      OUTPUT A
BE 00   MOV R6,#00
54 74   CALL WRITE
BE 3F   MOV R6,#3F
B9 3F   MOV R1,#3F      SET LOOP COUNTER
LOOP F1   MOV A,QR0      GET MEMORY DATA
54 74   CALL WRITE      MOVE TO EXT. MEM.
CE      DEC R6           DECR. POINTER
E9  --   DJNZ LOOP
93      RETR
```


Table 8: 8748 Debug Switch Settings

Function/Condition Desired	Switch 1 Ext Ram/ Int ROM	Switch 2 Single Step/ Run	Switch 3 Run/ Reset
Load RAM or Reset	Don't Care	Closed	Closed
Run w/External RAM	Open	Closed	Open
Run w/Internal ROM	Closed	Closed	Open
Single Step w/Ext RAM	Open	Open	Open
Single Step w/Int ROM	Closed	Open	Open

Table 9: Interface Cable Wiring

Signal	SBC	J1 Pin	Serial Board Connector
Data/Address Bit 7	7	2	8
"	6	4	7
"	5	6	6
"	4	8	5
"	3	10	4
"	2	12	3
"	1	14	2
"	0	16	1
Single Step Pulse		18	11
Addr Latch Enable		20	13
Write Enable		22	10
Chip Select		24	12
Address Bit 8		26	16
"	9	28	15
Ground		Odd #	9

Table 10: I/O Port Control Words

RAM Write:

Port E5: Mode 0 output	Control Word: 80 Port E7
Port E6: Mode 0 output	
Port E9: Mode 0 input	Control Word: 80 Port EB
Port EA: Mode 0 output	

Single Step:

Port E5: Mode 0 input	Control Word: 82 Port E7
Port E6: Mode 0 output	
Port E9: Mode 0 input	Control Word: 82 Port EB
Port EA: Mode 0 input	

BIBLIOGRAPHY

Naval Research Laboratory Report 8062, Fiber Optics for Naval Applications: An Assessment of Present and Near-Term Capabilities, by G. H. Sigel, Jr., 24 September 1976.

MIL-STD-1553A, Aircraft Internal Time Division Multiplex Data Bus, 30 April 1975.

Niemann, W., A Real-Time Operating System for Single Board Computer Based Distributed Naval Tactical Data Systems, M.S. Thesis, Naval Postgraduate School, June 1978.

Intel Corporation, MCS-48 Microcomputer User's Manual, 1978.

Rolander, T., Intel Multibus Interfacing, Application Note AP-28, Intel Corporation, 1977.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, VA 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, CA 93940	2
3. Department Chairman, Code 62 Department of Electrical Engineering Naval Postgraduate School Monterey, CA 93940	1
4. Professor Uno R. Kodres, Code 52 Kr Department of Computer Science Naval Postgraduate School Monterey, CA 93940	3
5. Richard Ernest Schue Naval Avionics Center, D/834 Indianapolis, IN 46218	2
6. Library Naval Avionics Center Indianapolis, IN 46218	2

17 SEP 81

26801

279861

Thesis

S355 Schue

c.1 A serial fiber optic
data bus for a distrib-
uted microcomputer sys-
tem.

17 SEP 81

26801

Thesis

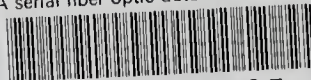
S355 Schue

c.1 A serial fiber optic
data bus for a distrib-
uted microcomputer sys-
tem.

279861

thesS355

A serial fiber optic data bus for a dist



3 2768 001 00343 7

DUDLEY KNOX LIBRARY